Newsgroup

1992	Subject	Author
07/20	What is literate programming, anyway?	Jim Glover
07/21	FWEB and debuggers	Jonathan Gilligan
07/21	Literate programming in the real world	Thorsten Ohl
07/22	FunnelWeb opinions	Joey Gibson
07/23	CACM articles on literate programming	Rob Beezer
07/31	A "Tempest in a Teapot"?	Cameron Smith
08/06	CWEB history	George Greenwade
08/07	CWEB formatting	Mike Yoder
09/18	First contact	Bryan Oakley
09/23	Renaming module	Joachim Schrod
12/09	How did tangle and weave get named?	Daniel Luecking
12/22	Object oriented literate programming?	Paul Lyon
12/29	Arachne	Jonathan Gilligan
1993	**Subject**	**Author**
01/04	Literate programming, why not routines instead?	Edward Keith
01/07	Defining multiple sections in one	Steve Avery
01/22	A programmer's first use of literate programming	Trevor Jenkins
02/22	Portable C/F/WEB	Philip Rubini
02/23	Alternatives to TeX and WEB	Glyn Normington
02/24	Literate progamming and Ada	Frank Pappas
03/31	Mini-indexes in TeX: The Program	Richard Walker
04/01	Literate Programming Using FrameMaker	Stephen Cross
04/08	Future developments	Anthony Coates
04/13	Experiences with literate programming so far	Kayvan Sylvan
04/14	Current view of literate programming	Vince Mehringer
04/21	Module decomposition advice	John Nicoll
04/23	Comments on simplicity	Bart Childs
04/30	Duplicate sections - good style	Lee Wittenberg
05/13	Anybody there?	Kayvan Sylvan
05/27	Unresolved sections in CWEB	Zdenek Wagner
06/01	Scraps with explicit arguments	Humberto Zuazaga
06/24	Are modules necessary?	Stephen McKearney
07/01	Transportable webs in SGML	Edward Keith
07/02	WEB in a business context	Dominique Dumont
07/14	Techniques for wide acceptance of literate programming?	Guy Bailey
07/15	WYSIWYG webs and debugging	Bryan Oakley
08/06	Publishing WEB programs	Mary Bos
08/26	Inverse comment convention	Dave Love
08/27	Code reordering	Lee Wittenberg

08/27	Little TeX needed	Uttam Narsu
08/27	Literate programming with troff or texinfo	Norman Ramsey
09/16	**comp.programming.literate passes**	Matthias Neeracher
09/18	Inline comments	Stuart Ferguson
09/21	What does literate programming mean to you?	Craig Hubley
09/22	Luminary abuse	Norman Ramsey
09/22	Multi lingual literate programming	Evan Kirshenbaum
09/23	Readable programs - an alternative to Web	Stephen Savitzky
09/23	Big programs	Robert McLay
09/27	LitProg Review	George Greenwade
09/29	Notation issues	Lee Wittenberg
10/18	Literary nomenclature	Joachim Schrod
10/20	Results of survey on prettyprinting	Conrado Martinez-Parra
10/25	Some general thoughts on literate programming	V. Chandrasekhar
11/18	Ordering dependencies and scoping	Zdenek Wagner
11/18	Reuse of literate programs	Mark Probert
12/03	CWEB thoughts	Phil Jensen
12/21	C++ API extractor	Paul Bandler
1994	**Subject**	**Author**
01/18	Importance of indexing?	Mark Carroll
01/18	Format of literate programming bibliography	Pat Pinchera
01/19	Literate hypertext indices	Mark Carroll
01/20	SGML and software development	Steve Heaney
01/21	Index for function definitions	Joachim Ziegler
01/25	Reengineering and literate programming	Jim Crigler
01/25	Usage of multiple-output files facilities	Lee Wittenberg
01/26	Novice looking for recommendations	Tom Epperly
01/28	Seeking K&R-style formatting for C	Wheeler Ruml
02/01	Make and noweb cpif	Lee Wittenberg
02/04	How do I use noweb effectively for C++?	Marcus Speh
02/04	Two points	Andreas Stirnemann
02/10	Numbering in CWEB	Greg Fiehler
02/11	Emacs mode for noweb	Karel Zuiderveld
02/11	How do you write C++ classes in CWEB?	Jerome Chan
03/03	Why do these questions keep popping up here?	Mary Bos
03/09	News from Phoenix (ACM conference)	Lee Wittenberg
03/10	Long array initializers in CWEB	Peter Jensen
03/15	Literary programming and C++	Anssi Porttikivi
03/22	Verbatim in CWEB?	Denis Roegel
03/23	Literate scripts in functional programming languages	Andrew Butterfield
03/24	Refinements	John Ramsdell
03/31	Seeking account of experiences with noweb	Norman Ramsey

04/06	Information on literate programming?	Greg Fiehler
04/19	Are there WEBs for visual languages (Visual Basic)?	Stephen Boyan
04/21	Suggestion for function prototypes in CWEB	Phil Jensen
04/22	How do I begin literate programming?	Eric Lemings
04/22	WEB system for Perl or Modula-2	Gregory Tucker-Kellogg
04/29	Macro/scrap definition within a context?	Mike Elliott
04/30	Macro preprocessing in language-independent WEBs	Gregory Tucker-Kellogg
05/05	Is there a CWEB for IBM BookManager?	Stephen Boyan
05/06	Printing CWEB title to an odd-numbered page?	Matt Pharr
05/08	TeX is uncompromising in its quality	Jeffrey McArthur
05/13	Writing portable code - any suggestions?	Felix Gartner
05/13	DVI specification and description	Brian Edginton
05/17	Who's using what?	Lode Leroy
05/27	Bad style in Stanford GraphBase	Tommy McGuire
06/09	Multiple module references	Robert Partington
06/09	C to LaTex converter	Graham Trigge
06/16	Reverse-engineering code	Tony Coates
06/19	CWEB formats C++ badly	Anssi Porttikivi
06/20	Can one have it all?	Allan Adler
06/21	FWEB vs. statement numbers	Allan Adler
06/21	FILE and LINE directives	Mark Naumann
07/12	How to define pretty-printing for variables like "N"?	Matthew Pharr
07/12	Operator overloading in CWEB	Greg Fiehler
07/15	Bold identifiers and plain keywords	John Scholes
07/29	Anyone use ProTex?	Sean Boyle
08/02	A parsing bug in CWEAVE?	Felix Gartner
08/31	Arguments for literate programming	Denis Roegel
09/16	Converting Pascal-WEB programs into C-CWEB	Andreas Scherer
09/24	Makefiles and configuration scripts in web files	Dietrich Kappe
10/03	Cross-referencing in CWEB	Charles Blair
10/04	Wishes for CWEB features	Yotam Medini
10/12	Not-small C program with noweb	Eric Prestemon
10/15	Empty space in noweb TeX output	Joerg Viola
10/17	Style of header file inclusion	Felix Gaertner
10/17	Several document types from the same noweb file?	Fernando Mato Mira
10/19	WEB for Turbo Pascal	Denis Roegel
10/21	C/C++ -> TeXinfo/HTML/LaTeX/nroff -man	Oliver Imbusch
10/21	What is cpif?	Sven Utcke
10/26	How do you write (small) classes with CWEB?	Giovanni Pensa
11/04	Header file creation in CWEB	Ender Olcayto
11/11	WEB TeX to MWeb TeX to OWeb TeX	Jeffrey McArthur
11/18	CWEAVE and MFC class wizard	Dave Hamilton

11/28	Proposal: OpenDoc/OLE literate programming tool	Tony Coates
11/30	Usage of CWEB	Yuval Peduel
12/10	Making noweb produce ASCII text	Joseph Brothers
12/19	How to distinguish x from x.* in indices (noweb)	Balasubramanian Narasimhan
1995	**Subject**	**Author**
01/01	Compliments to the developers of noweb and nuweb!	Ben-Kiki Oren
01/11	CWEB: Output without c-code	Claudia Hattensperger
01/24	Newbie question (not from AOL)	Eric Dahlman
01/30	Separate compilation	Ben-Kiki Oren
02/01	Clueless literate programming questions are welcome	Norman Ramsey
02/01	Spidery WEB for Perl?	Diego Zamboni
02/12	Are verbose comments appropriate?	Mel O Cinneide
02/18	Is this a good style?	Darrell Grainger
02/22	Current status of NUWEB?	Stephen Simmons
03/07	noweb: proof of the pudding	Ozan Yigit
03/08	Meta literate programming	Mark Kramer
03/13	Names of 'chunks'	Robert Partington
03/20	An extended web language	Carsten Tinggard Nielsen
03/29	Bizarre and cryptic language	Will Ware
04/18	WEB++	Michal Gomulinski
04/30	C++ templates and FWEB	Come Raczy
05/14	Is literate programming useful for object-oriented programming?	Jacob Nielsen
05/14	My experience with literate programming	Greg Humphreys
05/14	Moderate comp.programming.literate	Will Ware
05/17	Beginner's guide?	Kumaran Santhanam
05/18	Why is literate programming nearly unused?	Basile Starynkevitch
05/19	Web for lex/yacc	Don Hosek
05/22	SPIDER generated WEB for Lisp?	Paolo Amoroso
05/24	Perfect literate programming tool gripe	Dietrich Kappe
05/25	The development cycle and literate programming	Paolo Amoroso
05/28	Web systems and TeXinfo?	Mike Eggleston
06/17	Encapsulation in literate programming	Kristopher Johnson
06/19	Why doesn't CWEB have true comments?	John Bay
07/20	Literate programs for Microsoft Windows applications	Steve Furlong
08/17	Using smaller margins in noweb?	Robert McLay
08/20	Literate programming GUI?	Anthony Yen
08/21	Literate programming in industry	Jim Sisul
08/23	CWEB and C++ templates	Stefan Thienel
08/24	Literate programming without TeX	Keith Ballinger
09/04	C(++) to WEB translation?	Gilbert van den Dobbelsteen
09/05	CWEB cross references to sections	Kendall Shaw
09/29	Some questions on noweb	Alberto Meroni

09/29	Program prettyprinting macros	Carlos Felippa
10/13	Publishing forums	Kiyoshi Akima
11/05	CWEB & C++ trouble with `const'	Jan Dvorak

Announcements

Date	Subject	Author
1992	Announcements for MetaPost	John Hobby
1993	Announcements for CLiP	Eric van Ammers
1993	Announcements for noweb	Norman Ramsey
1993	Announcements for WinWord WEB	Lee Wittenberg
1993	Announcements for nuweb	Preston Briggs
1993	Announcements for FWEB	John Krommes
1993	Announcements for CWEBx	Marc van Leeuwen

What is literate programming, anyway?

From: **Jim Glover**
Date: 20 Jul 1992

Right off the bat, I have a question. Now that I have subscribed, what is literate programming, anyway? I'm serious here. I don't know what it means, but it sounds interesting. Now, let me ask your forgiveness for this one: To branch, or not to branch? That is the question.

From: **George Greenwade**
Date: 20 Jul 1992

I'm truly sorry to post such poor reply to the first message, but approximately 20 more people are on-line since this was posted originally. Jim Glover asked:

Right off the bat, I have a question. Now that I have subscribed, what is literate programming, anyway? I'm serious here. I don't know what it means, but it sounds interesting. Now, let me ask your forgiveness for this one: To branch, or not to branch? That is the question.

I, too, would truly appreciate a semi-concise definition. I know it has to do with portability, the ability of a language to port itself to another language, etc., but might someone be willing to provide the answer to what I am sure will be a FAQ for new subscribers and uninitiates, such as myself? Cameron? Don? Anyone else? Regards and thanks for your interest in this list, George.

From: **Cameron Smith**
Date: 20 Jul 1992

Jim Glover asks what literate programming is.

In Don Knuth's 1984 article "Literate Programming", in which he introduced the term, he indicated that he chose the name "literate programming" in part to contrast with "structured programming", which he apparently felt had the wrong orientation. He says: "Let us change our traditional attitude to the construction of programs. Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do." He goes on to discuss programs as literature, written for human beings to understand. He introduces WEB as a tool to assist in the secondary task of massaging human-readable programs into a form that a computer can execute.

The above-referenced article, together with a lot of other writings about styles and standards for reliable software engineering, is collected in an anthology: Literate Programming, by Donald E. Knuth CSLI Lecture Notes Number 27 copyright 1992, Center for the Study of Language and Information, Leland Stanford Junior University LC catalog number QA76.6.K644 1991 ISBN

0-9370-7380-6(paper), 0-9370-7381-4 (cloth). I paid $24.95 for my (paperbound) copy. I recommend it enthusiastically.

Jim also asks: To branch, or not to branch? That is the question.

The longest single chapter in the above-cited book is "Structured Programming with go to Statements"; it includes a bibliography of 103 entries surveying the literature on the subject at the time of its writing (1974). Considering that this article is nearly 20 years old now, I was surprised at how much I learned from reading it. In particular it was the first time I had heard of the "situation" construct for governing flow of control.

FWEB and debuggers

From: **Jonathan M. Gilligan**
Date: 21 Jul 1992

I have used CWEB a bit, but am dissatisfied with the difficulties I've found using it with modular programs. I plan to try using Don Hosek's suggestions for implementing prototypes and header files, but I'm also curious about FWEB. Is FWEB small enough to compile on a PC (i.e. with 64k segments and about 600K available memory)?

I am just learning C++ and it seems to me that designing class hierarchies is an extreme case of something that needs literate programming. I am now spending more time sitting with a pencil and paper sketching data structures and relations than thinking about code and it's clear to me that it will not be possible to keep track of what's what without taking a literate approach.

I've been put off by the fact that every WEB program I've read starts out with the same sort of I/O code to translate between external and internal character sets (TeX, MF, bibTeX, etc.) where it would be far more efficient to have an I/O package that you could just reference. This is along the lines of the criticism of DEK's WEB style in Programming Pearls ten years ago or so, that DEK treats a WEB program as though there were no libraries or wheels in the world.

Also, I'd be very interested to hear how people deal with web files and symbolic debuggers, particularly in a DOS environment. One of the constraints that has kept me from using WEB extensively is the difficulty of getting a symbolic debugger to associate the code with the right part of the .web source file. This is exacerbated by the fact that when TANGLE is done with a file, it's pretty much unreadable and hence of little use for symbolic debugging.

Finally, when writing multimodule webs, it would be desirable to include things like makefiles and module definition files in the web --- after all I consider these to be integral parts of the code, since I need them to compile it and the makefile is usually the place where we document best what the dependencies are. What are people's thoughts on these issues?

From: **Thorsten Ohl**
Date: 21 Jul 1992

Jonathan M. Gilligan said: [...] but I'm also curious about FWEB. Is FWEB small enough to compile on a PC (i.e. with 64k segments and about 600K available memory)?

Yes. It is known to compile with Micro$oft C 6.00 (once upon a time, it also compiled with Turbo C, but I never tried that).

Also, I'd be very interested to hear how people deal with web files and symbolic debuggers, particularly in a DOS environment. One of the constraints that has kept me from using WEB extensively is the difficulty of getting a symbolic debugger to associate the code with the right part of the .web source file.

FWEB inserts sync lines `#line 137 foo.web' into the code, so any compiler/debugger worth its money should respect them. And I seem to remember from my MS-DOS days that it worked ...

This is exacerbated by the fact that when TANGLE is done with a file, it's pretty much unreadable and hence of little use for symbolic debugging.

There's another proplem: how to distribute literate sources to illiterate users? Many people are not (yet?) willing to use the .web sources, but are also slightly offended by tangled code (if they want to make modifications). Yes, I know, they should be taught to use change files ...

Finally, when writing multimodule webs, it would be desirable to include things like makefiles and module definition files in the web --- after all I consider these to be integral parts of the code, since I need them to compile it and the makefile is usually the place where we document best what the dependencies are.

FWEB plans to support Makefiles sometime in the future. The problem remains that you easily end up remaking everthing, once your .web file has changed. It would be nice if tangle/weave could operate on multiple files, such that the documentation (including the index!) would be one single document, while the code could be compiled separately.

From: **John Fieber**
Date: 21 Jul 1992

Also, I'd be very interested to hear how people deal with web files and symbolic debuggers, particularly in a DOS environment. One of the constraints that has kept me from using WEB extensively is the difficulty of getting a symbolic debugger to associate the code with the right part of the .web source file. This is exacerbated by the fact that when TANGLE is done with a file, it's pretty much unreadable and hence of little use for symbolic debugging.

While this is an important issue that deserves serious attention, I would like to add that since I've started using CWEB, my debugger has just been collecting dust. Seems as though Knuth was right about well thought out code needing less time debugging. Web inspires well thought out code. It works for me. :) Out of curiosity, what sort of platforms are people using the various incarnations of web on? Personally I'm using CWEB on an Amiga. I have FWEB kicking around but have not even unpacked the archive yet.

From: **Fritz Zaucker**
Date: 21 Jul 1992

There's another proplem: how to distribute literate sources to illiterate users? Many people are not (yet?) willing to use the .web sources, but are also slightly offended by tangled code (if they want to make modifications). Yes, I know, they should be taught to use change files ...

Hmmm, after doing an GNU EMACS indent-region on my ctangled file it actually looked pretty much ok to me.

From: **Don Hosek**
Date: 21 Jul 1992

Also, I'd be very interested to hear how people deal with web files and symbolic debuggers, particularly in a DOS environment. One of the constraints that has kept me from using WEB extensively is the difficulty of getting a symbolic debugger to associate the code with the right part of the .web source file. This is exacerbated by the fact that when TANGLE is done with a file, it's pretty much unreadable and hence of little use for symbolic debugging.

With CWEB and the Zortech debugger, I have seamless debugging. It always looks in the right place in the .w file for the line being executed. (zdb has other problems but...). On the other hand, the Turbo Debugger seems to ignore the file names on #line statements so it's useless in CWEB (if you aren't pointing at another file, why even have #line statements?). Of course pascal-weavers (my nominee for what we call ourselves) are completely out of luck having no equivalent to #line.

From: **Cameron Smith**
Date: 21 Jul 1992

Thorsten Ohl writes: There's another proplem: how to distribute literate sources to illiterate users? Many people are not (yet?) willing to use the .web sources, but are also slightly offended by tangled code (if they want to make modifications).

Actually, I started using c-no-web over a year before I first began to dabble in CWEB, for the same reasons that Jim Fox created it: I wanted to write well-documented, attractively typeset programs but I didn't want to (a) adopt an entirely new philosophy of programming or (b) spend time acquiring arcane skills with yet another Knuthianly quirky and idiosyncratic software system. (I think the man is brilliant but demented -- the macro-expansion semantics of TeX and Metafont are intellectually interesting, but they are hell to use for actual programming.)

I see the chief value of literate programming as being the care that one is forced to take over one's code when one keeps in mind while writing it the goal of making it understandable to others. I know that the thought that someone else might see what I was doing discourages me from pulling dirty tricks and shortcuts in my code!

But any idea that WEB lets one develop the parts of one's programs in the order that makes the most sense is just silly. My text editor lets me move up and down in my source files, thank you very much, so I can work on the parts of the code in any order I want to. The value of WEB specifically (considered as one particular tool but not the only possible tool for literate programming) is that it lets you present the program, after it is written, in the order that it was developed -- or in the order that makes the best presentation, which is probably NOT the order in which it was developed.

So I ask myself: does that advantage outweigh the disadvantages (such as not being able to share source files with non-WEBsters, and having headaches with Makefile dependencies)? And for many projects the answer is "no, it doesn't". For those I greatly prefer c-no-web, which still lets me create attractive, legible program listings, including illustrations, mathematical derivations, or whatnot, but doesn't require me to wangle (or mangle or whatever) the source file before I can TeX it or compile it or share it with a colleague. (For those who haven't tried it, let me explain: a c-no-web file is acceptable as input to both TeX and a C compiler without any preprocessing!) For short programs especially, ones that a reader wouldn't really need an outline or index or roadmap to find his way around in anyway, c-no-web is the way to go. (It does let you have sections, subsections, a table of contents, and other nice features, but it doesn't do the cross-referencing that WEB-based systems do.)

I might also mention the fact that c-no-web doesn't impose someone else's style of indentation on me (there are some features of the way CWEB treats C code that I hate), and its TeX macros are much easier to customize than CWEB's. (For a system whose sole purpose is to encourage clean style and good documentation in programming, CWEB uses inexcusably poorly documented TeX code, and some "features" of its typography can only be fixed by changing the grammatical productions it uses and recompiling cweave.)

I realize that the above criticisms go to the existing implementation of CWEB and not to the underlying ideas. It's just offered as a quick take on my (admittedly limited) experience with these two approaches as they now exist, and doesn't in any way imply that a WEB-like system without these problems couldn't be created and be a useful tool. (In fact I do sometimes use CWEB, but I sometimes grit my teeth when I do.) I also realize that c-no-web doesn't help people who program in other languages than C.

From: **Dave Love**
Date: 22 Jul 1992

Thorsten Ohl said: There's another proplem: how to distribute literate sources to illiterate users? Many people are not (yet?) willing to use the .web sources, but are also slightly offended by tangled code (if they want to make modifications). Yes, I know, they should be taught to use change files ...

There's a system called `noweb' by Norman Ramsey (of SpiderWeb fame) which addresses problems of interacting with make and tangling code to a form that can be sensibly exported to the `illiterate' :-). It's unix-based (modular) and doesn't pretty-print, although you could write pretty-printing modules for it and, presumably, adapt the shell scripts to other operating systems with more or less difficulty.

Literate programming in the real world

From: **Thorsten Ohl**
Date: 21 Jul 1992

There's another proplem: how to distribute literate sources to illiterate users? Many people are not (yet?) willing to use the .web sources, but are also slightly offended by tangled code (if they want to make modifications). Yes, I know, they should be taught to use change files ...

Fritz Zaucker said: Hmmm, after doing an GNU EMACS indent-region on my ctangled file it actually looked pretty much ok to me.

Sure, that solves some of the problems. And the following make(1) rule gives me a FORTRAN source that's almost indistinguishable from some standard FORTRAN coding conventions :-). [It still has too much indentation though ...].

```
CLOV.FORTRAN: clov.f
sed -e '/^[Cc\*]/d' -e '/^[ ]*$$/d' -e '/^[ ]*CONTINUE/d' $< \ | tr a-z A-Z > $@
```

Still, all the comments are gone and the source is therefore almost useless.

I'm really curious about your experiences with *web in the Real World(tm). I have to confess that I frequently prototype code in FWEB, but when it comes to creating the production version, I surrender to my conservative colleagues (Physicists can be _very_ conservative :-{} and use ... umm, err, umm .. Fortran. How do you convince your illiterate colleagues of the merits of literate programming?

PS: But I should really try to submit a FWEB program to Comp. Phys. Comm. one day ...

From: **Charles Elliott**
Date: 22 Jul 1992

Would someone please post a brief paragraph on 'literate' programming for us novices? Many of us, I am sure, are so pressed for time that we don't really have the time to devote to a scholarly article, a tutorial or independent investigation...just to see if 'literate' programming is worth serious attention. BTW the term 'illiterate' is one of opprobrium in normal use, and I'll bet other neophytes are offended by it as well. Can't you think of a better term? 'unconvinced'?

From: **Cameron Smith**
Date: 22 Jul 1992

Charles Elliott asks: Would someone please post a brief paragraph on 'literate' programming for us novices? [...]

Literate programming is writing programs for people rather than machines. (How's that for brief? :-))

The term was chosen (in part) to suggest going beyond "structured programming", which means programs written in a good way for machines to understand and execute, into a mode in which programs are written primarily for humans to understand and appreciate. The grungy details of massaging a human-readable exposition of an algorithm into something that a machine can digest are considered secondary, and tools like WEB are supposed to take care of that for us.

Literate programming means regarding programs as contributions to the literature of computer science. Knuth firmly believes that we write better programs when we address them to our peers than when we address them to our tools. He maintains that the discipline required to write programs in a way that would let others understand them, with clean exposition and attractive presentation, automatically causes us to write better programs, as a by-product, while ennobling the craft by keeping its focus directed at people rather than machines. Literate programmers buy this idea.

[...] the term 'illiterate' is one of opprobrium in normal use, and I'll bet other neophytes are offended by it as well.

This is intentional (not that offense be taken, but that 'illiterate' be a term of opprobrium). Donald Knuth explicitly stated that another factor in his choice of terminology was the intention of making programmers feel embarrassed to admit that they write 'illiterate' programs. Maybe this isn't good psychology or good PR, but it's not accidental. And since those who practice literate programming presumably do so because they do believe that it makes better programs, they probably don't mind expressing disapprobation of lesser methods (just as programmers who religiously write structured programs sneer at "spaghetti code"). So I doubt it will change.

From: **Jonathan M. Gilligan**
Date: 22 Jul 1992

Charles Elliott writes: Would someone please post a brief paragraph on 'literate' programming for us novices? Many of us, I am sure, are so pressed for time that we don't really have the time to devote to a scholarly article, a tutorial or independent investigation...just to see if 'literate' programming is worth serious attention.

For a short discussion, see the book review of Knuth's Stanford Tech. Report, ``Literate Programming,'' in this Month's (August 1992) Dr. Dobb's Journal. The review is somewhat incomplete, mostly in that it does not touch on some of the shortcomings of existing literate programming systems, does not acknowledge the existence of systems other than WEB and CWEB, and doesn't mention any literature other than the Tech Report, but it gives a reasonable flavor of what the basic ideas are and it is easy to read. A much more thorough discussion, which is more involved (but you can read it easily and catch the main ideas even if you skip the hard parts) is the two-part Programming Pearls column in CACM, Vol. 29, pp. 364--369 (May, 1986) and 471--483 (June, 1986).

If you want a description short enough to post, I can't improve on Cameron Smith's.

Jim Glover wants examples of literate code. Again, the Programming Pearls piece is a nice place to start. It has a short and a not-so-short example, along with some very thoughtful criticism of literate programming and Knuth's programming style by Doug McIlroy of Bell labs. There are also a number of WEB programs on labrea.stanford.edu, which you can ftp anonymously.

From: **Anders Thulin**
Date: 23 Jul 1992

Timoty Murphy wrote: @book{knuth92 author = {Donald E. Knuth}, title = {Literate Programming}, publisher = {CSLI}, address = {Stanford}, year = {1992}}

Is this book/report any good? Seriously?

When I first read about literate programming, I was rather impressed by the idea. However, when I tried to read some of the Pascal WEB's that Knuth wrote I was very disappointed. The only major WEBs I've studied (the TeX and METAFONT source) are not well adapted to human understanding: they present the pieces bottom up instead of top down, which would be more natural for most readers.

Rather than going beyond the program structure imposed by the Pascal syntax, they submit to it, so that the TeXl WEB is not more easily read than an ordinary Pascal program: all the little pieces come first, and the main program is kept a secret till the last few lines, much like the verb in German sentences. The only difference from straight Pascal code is that there is some descriptive text - but that helps only micro-understanding of the program rather than macro-understanding: the module structure, the flow of data, etc.

I don't doubt that literate programming can be good - I've just not seen example where Knuth demonstrates it convincingly. Now: is this reference an improvement? Alternatively, are there any good literate programs out there? Some that really show what's literate programming is about?

I'd like to add to the references on literate programming: I recall a series on literate programming in the ACM several years ago. I can't find them again, so it must have been several years ago. In each article, as I recall it, a literate program by some author (Knuth, Gries, Jackson?) was presented and then someone else criticised it. I think these articles gave a pretty good insight into the mechanism of writing a literate program as well as that of reading one. If anyone can give a more definite reference to these articles, I would appreciate it.

From: **Cameron Smith**
Date: 23 Jul 1992

Anders Thulin writes: I recall a series on literate programming in the ACM several years ago. I can't find them again, so it must have been several years ago. In each article, as I recall it, a literate program by some author (Knuth, Gries, Jackson?) was presented and then someone else criticised it. I think these articles gave a pretty good insight into the mechanism of writing a literate program as well as that of reading one.

I think the CACM series you're talking about is the one that Knuth alludes to in the preface to the Literate Programming anthology that has already been cited so often in this forum. If so, the editor of that series was (according to Knuth) Chris Van Wyk, and it began not too long (he isn't specific) after the Programming Pearls columns of May and June 1986, in which Jon Bentley wrote about his impressions of literate programming and presented two Web programs of Knuth's and a critique of the longer one by Doug McIlroy. I've been meaning to get over to the library and chase them down; maybe I can do that this afternoon, and if so I'll post citations (unless someone else does it first).

Is this book/report any good? Seriously? When I first read about literate programming, I was rather impressed by the idea. However, when I tried to read some of the Pascal WEB's that Knuth wrote I was very disappointed. [...] I don't doubt that literate programming can be good - I've just not seen example where Knuth demonstrates it convincingly. Now: is this reference an improvement?

The book is an anthology of writings of Knuth's published over a 20-year period. It is interesting to read in part because it gives you an impression of how a brilliant man's ideas evolved, which I think gives less gifted guys like me insights that we wouldn't perhaps be able to achieve on our own. Does it say anything that hasn't been said before? Of course not, by definition, because it's an anthology of past works. For me, the two most enlightening parts were the article "Structured Programming with go to Statements", which antedates Literate Programming by 5-10 years (it came out in 1974, five years before Knuth developed WEB and almost exactly 10 years before the "Literate Programming" article appeared), and McIlroy's incisive, balanced and erudite critique of Knuth's literate program in the Pearls column (which McIlroy praised as a piece of expository writing but which, when considered as a piece of engineering, he called "an industrial-strength Faberg\'e egg---intricate, wonderfully worked, refined beyond all ordinary desires, a museum piece from the start"). I also found the chapter on the errors of TeX valuable; it's an insightful and reflective survey of the problems encountered in a huge software engineering project. I happen to think one case study is worth a trunkful of textbooks, especially this one, because he published not only his post facto evaluation of what transpired, but also his log book, so you get a real feeling for how the project progressed.

At any rate, here's a list of the chapters and their dates of original appearance; you can track 'em down separately or buy the book (or neither). Computer Programming as an Art 1974 (This was Knuth's Turing Award lecture). Structured Programming with go to Statements 1974. A Structured Program to Generate All Topological Sorting Arrangements 1974. Literate Programming 1984. Programming Pearls: Sampling May, 1986. Programming Pearls, Continued: Common Words June, 1986. How to Read a WEB 1986 (This is only 6 pages long, and is completely redundant with other material in the book. I have no idea why it was included, unless maybe to help out people who might want to read the TeX and Metafont excerpts without having read the preceding 3 chapters). Excerpts from the Programs for TeX and Metafont 1986. Mathematical Writing 1987 (This is an excerpt from course notes; you can get the full monograph from the AMS, but most of the rest has little to do with programming). The Errors of TeX 1989. The Error Log of TeX 1978-1991 (This is the complete log as of September 1991, including some hitherto unpublished material thought lost but recently discovered in the Stanford library archives). An Example of CWEB 1990 (This program, written with Silvio Levy, is the "wc" example

distributed with CWEB.).

Is it worth it? I thought so, but if you aren't a TeX junkie and you already understand what literate programming is about, maybe not.

From: **Don Hosek**
Date: 23 Jul 1992

When I first read about literate programming, I was rather impressed by the idea. However, when I tried to read some of the Pascal WEB's that Knuth wrote I was very disappointed. The only major WEBs I've studied (the TeX and METAFONT source) are not well adapted to human understanding: they present the pieces bottom up instead of top down, which would be more natural for most readers.

Rather than going beyond the program structure imposed by the Pascal syntax, they submit to it, so that the TeXl WEB is not more easily read than an ordinary Pascal program: all the little pieces come first, and the main program is kept a secret till the last few lines, much like the verb in German sentences. The only difference from straight Pascal code is that there is some descriptive text - but that helps only micro-understanding of the program rather than macro-understanding: the module structure, the flow of data, etc.

My first WEB was DVIview, a previewer for VM/CMS in which I followed a structure more like what was desired. The outline was borrowed from somewhere else, but to summarize, the only @p section read something like:

```
@p
program Dviview;
@<Constant definitions@>;
@<Global variables@>;
@<Forward function and procedure references@>;
@<Global functions and procedures@>;
begin
@<Initializations@>;
@<First step@>;
@<Second step@>;
@<Etc.@>;
end.
```

Then each piece was exploded as appropriate. Aha... found it. A nice concise, well-written web: primes.web (should be distributed with every Pascal WEB distribution). Not quite the scheme above (not exactly necessary in this case), but close enough.

From: **Timothy Murphy**
Date: 23 Jul 1992

Rather than going beyond the program structure imposed by the Pascal syntax, they submit to it, so that the TeXl WEB is not more easily read than an ordinary Pascal program: all the little pieces come first, and the main program is kept a secret till the last few lines, much like the verb in German sentences.

As I'm sure everyone on this list will know, this was a decision of Knuth's -- right or wrong -- and not a consequence of the WEB format, which allows the main program to come at the beginning or at the end.

From: **Anders Thulin**
Date: 24 Jul 1992

Rather than going beyond the program structure imposed by the Pascal syntax, they submit to it, so that the TeXl WEB is not more easily read than an ordinary Pascal program: [...]

Timothy Murphy writes as a reply to my earlier post: As I'm sure everyone on this list will know, this was a decision of Knuth's -- right or wrong -- and not a consequence of the WEB format, which allows the main program to come at the beginning or at the end.

Of course. But as always, people seem to learn by example. I've seen a few attempts at literary programming in Pascal, and all used exactly the same approach as in the TeXbook: main program last. When I asked why, none of the authors knew exactly why. It just turned out that way - I can only assume that it somehow is an artifact of the coding process. But since it followed the pattern of Knuth's texts, nobody thought there was anything very much wrong with it.

And that is, I think, the point I'm trying to make. A literary program should be written to mirror how someone learns about a new and

complex system - if necessary by lying a little (see the TeXbook). It should be designed, rather than just grow. Those literate programs I've read so far do not show any clear evidence of design, or any discernible effort of trying to present things in the right order for learning.

That's why I asked: are there any good literary programs around? - programs that can serve as examples of the benefits of literary programming as well as patterns to model ones own attempts at literate programming on. I think that giving one of those to people who ask abut literate programming will do much to demonstrate the utility of it all.

From: Donald N. Petcher
Date: 24 Jul 1992

As I'm sure everyone on this list will know, this was a decision of Knuth's -- right or wrong -- and not a consequence of the WEB format, which allows the main program to come at the beginning or at the end.

Anders Thulin writes: Of course. But as always, people seem to learn by example. I've seen a few attempts at literary programming in Pascal, and all used exactly the same approach as in the TeXbook: main program last. When I asked why, none of the authors knew exactly why. It just turned out that way - I can only assume that it somehow is an artifact of the coding process. But since it followed the pattern of Knuth's texts, nobody thought there was anything very much wrong with it.

A literary program first of all is governed by personal taste as to what is readable and enjoyable. For example, even though I have seen many C programs that put the main program at the end, I have never been tempted to do so. The way I think when I program dictates that I put the main program first, whether I am programming in C or web, and when I read my own programs at a later stage, this also seems intuitive to me. Probably some others just like to start reading details and get to the main story later, and still others like to put the program in the editor and hit the "Go to end of file" key to start reading the program and perhaps also turn to the back of the book first when reading the output. (After all, some like philosophy, some like science, some like mystery novels, and some like all three depending on the mood!) I still prefer the overview to be the first thing I see. (By the same token, I find it very frustrating reading Knuth's books.)

That's why I asked: are there any good literary programs around? - programs that can serve as examples of the benefits of literary programming as well as patterns to model ones own attempts at literate programming on. I think that giving one of those to people who ask abut literate programming will do much to demonstrate the utility of it all.

Although web has been around for quite awhile, it has not had such a following that these concepts are really fleshed out yet. We have some idea what style is for writing books, but for writing programs we still have a lot to learn and discuss. I anticipate that this group is a beginning and hope strides will be made to move toward a more uniform perspective on what tools we need to accomplish the goal, as well as an appreciation for different styles of good literary programs. (Speaking of tools, the issue is not settled from a programmer's standpoint either, as differences in philosophy are already reflected in the various versions of web around: e.g. FWEB vs. funnel web.)

Until some clearer consensus evolves, perhaps the best we can do is submit some of our own efforts, and collectively judge what we consider good examples of literate programming. Having said that though, I would venture to guess that if I looked back over my own programs I would not feel very confident about submitting any one as my own conception of what literate programming ought to be. I don't think I have ever polished one up to that extent. Just as in writing a research paper or a novel, there are degrees along the way to a fully finished and publishable product. To that extent, I guess the major examples remain the programs TeX and Metafont, as reflecting the judgements of Knuth. After all, he did publish them.

From: Marcus Speh
Date: 24 Jul 1992

Anders Thulin said: A literary program should be written to mirror how someone learns about a new and complex system - if necessary by lying a little (see the TeXbook). It should be designed, rather than just grow. Those literate programs I've read so far do not show any clear evidence of design, or any discernible effort of trying to present things in the right order for learning.

I think this is too restricted a use of the WEB system. I like the idea nevertheless, but: If WEB could only be used for pedagogic purposes, it could only serve as an individual toy. It is hard to convince my collaborators to make the effort to try to understand my woven files; they will only do so if I tell them: "Listen, this tool is going to increase the productivity of our group". People will have to worry less if someone else passes them a started program, or a program to be maintained. Most code "just grows" in the first place, and I am never going to redo all that work.

Note: I am not talking about a company, but about relatively large lattice field theory projects at our institute which involve frequently changing people. The pure idea of learning any new computing tool drives most of these guys mad. This is a consequence of the (still) low reputation of CS among (theoretical) physicists.

Might be I misunderstood you, Anders. I do not think there is one and only "good style" in literate programming (besides the usual programming sins which are largely eliminated in literate programming anyway). I think that the value of literate programming code will always be dictated by the kind of application you are using it for.

That's why I asked: are there any good literary programs around? - programs that can serve as examples of the benefits of literary programming as well as patterns to model ones own attempts at literate programming on. I think that giving one of those to people who ask abut literate programming will do much to demonstrate the utility of it all.

I second this point of yours! Though: There might be "good" literate programs for demonstrating how the literate programmer learnt to implement something, and there might be others which serve "group production" purposes. A third reason somebody else told me about (who is using WEB for years) is simply to have a nice documentation once he looks at old code again. It is clear that these literate programs may very well violate any "good style". I guess that I have been trying hard to express with my bad English what is trivially true for any meta-language [or however the experts call systems like WEB].

Summarizing, I would rather reject any proposal for a "good style" on top of literate programming, because I want to use WEB in order to do many things, namely

- rewrite old code [often by others] in order to better understand it
- make code better understandable for others so that I can give it away without bad conscience,
- provide documentation for myself in case I am still in this business some years from now ;-(
- satisfy my own sick need for writing something I consider beautiful (my most objective goal *-).

I should say that I only used FWEB so far, but the philosophy is the same everywhere, I assume.

From: **Jonathan M. Gilligan**
Date: 24 Jul 1992

A literary program should be written to mirror how someone learns about a new and complex system - if necessary by lying a little (see the TeXbook). It should be designed, rather than just grow. Those literate programs I've read so far do not show any clear evidence of design, or any discernible effort of trying to present things in the right order for learning.

Marcus Speh writes: I think this is too restricted a use of the WEB system. I like the idea nevertheless, but: If WEB could only be used for pedagogic purposes, it could only serve as an individual toy. It is hard to convince my collaborators to make the effort to try to understand my woven files; they will only do so if I tell them: "Listen, this tool is going to increase the productivity of our group". People will have to worry less if they someone else passes them a started program, or a program to be maintained. Most code "just grows" in the first place, and I am never going to redo all that work.

But Anders's point is applicable here. Read TeX.web and try to get a feel for how the program runs. First you get all the global data and initialization, then you go through I/O character-set translation, etc. The main routine that links the parts together is the last thing you see.

Nowhere is there an overview of what's going on or how the various phases of processing are invoked. You are drawn inexorably into the microscopic, despite what Knuth has said about a talent for programming being a talent for switching context rapidly between the forest and the trees. A good web should begin with an overview of what the program does and how the task is broken into chunks. This is important not just for pedagogy, but for the new person on the project, who needs to learn what this colossal piece of code written by her predecessors does.

Here, I find myself thinking of (and agreeing with) Doug McIlroy's complaint (CACM Vol. 29, pp. 471--483 (1986)) that WEB-style literate programming would be much better if there were facilities to include diagrams to help explain what's going on.

From: **Bradford Clark**
Date: 24 Jul 1992

Jonathan M. Gilligan writes: A good web should begin with an overview of what the program does and how the task is broken into chunks. This is important not just for pedagogy, but for the new person on the project, who needs to learn what this colossal piece of code written by her predecessors does.

Ditto, ditto. I always start my WEB code off with a explaination that will put the following program in context. What I would really love is to be able to talk to drawings included as part of the text file. A picture is worth a thousand words.

FunnelWeb opinions

From: **Joey Gibson**
Date: 22 Jul 1992

I have just recently begun delving into literate programming, using FunnelWeb. It has a very nice tutorial introduction, but are there any other introductory texts, such as "Essential LaTeX" or "TeX on a VaX"? I have recently received FunnelWeb and was wondering what others thought of this package. It is the first Web implementation I have used, and I rather like it, inasmuch as I know about Web, thus far. Anyway, opinions on FunnelWeb would be appreciated.

From: **Paul Lyon**
Date: 12 Dec 1992

I have done a port of FunnelWeb to OS/2 2.0. I used the _other_ port of gcc to OS/2 2.0 (yes, there are two of them!), namely gcc/2. That does not have DOS extender capability, so the executable will only run under OS/2 2.0. If there is any interest, this could be made available. (I plan to upload it to ftp-os2.nmsu.edu fairly soon, anyway.)

Has anyone else used FunnelWeb? I quite like it, perhaps in part because Ross Williams did such a fine job of it. Indeed, it may be worth acquiring just for that reason. It is as good as anything I have ever seen, and could serve as a model of how to put a modest size software package together.

From: **Paul Lyon**
Date: 16 Dec 1992

FWEB and FunnelWeb are quite distinct. FWEB is built on top of the CWEB framework; although the parser in its weave processor can do more---all of ratfor, C, C++, and (though the support is not complete), TeX, it is still confined to those specific languages, and still imposes on the user the formatting conventions that please its author. To get something different you will have to hack the weave processor. (One could possibly get somewhere by modifying the TeX macro package that FWEB uses, but that might be the harder way to go, unless, of course, you are already a TeXpert :-)

FunnelWeb, on the other hand, does not try to parse the "source" code at all; it just takes the layout of the source as written, turns off the meaning of plain TeX's special characters, sets typewriter font, and then invokes \obeylines and \obeyspaces; all this together causes TeX to print the source verbatim (the paragraph formatting is turned off, and TeX does not gobble spaces). The original WEB, CWEB, FWEB, and SpiderWeb all parse and format the source, inserting TeX math codes for the operators, putting keywords in boldfont, adjusting the indentation, and so on. If you like the style chosen for the programme, it looks much nicer that way. Except for SpiderWeb, which can be adapted to various languages by allowing a fair range of variation in specifying the pretty printing grammar using a large awk script to process the grammar spec and generate replacement code for significant chunks of weave, the pretty printing parser(s) in the other are hard coded. This has it uses besides making the typeset code more attractive; WEB, CWEB, FWEB, and SpiderWeb all do an index of identifiers for the code that can differentiate, for the most part, between declaration and use of an identifier (they know enough about the grammar to do that, but not, of course, as much as a compiler or interpreter).

None of the tangles (FunnelWeb included) distinguishes code from documentation by seeing what is TeX and what is not. Instead they all use special indicators for the one and the other. In FunnelWeb, e.g., anything enclosed in a `@{', `@}' pair is "source", anything without is either documentation or other directives to its tangle and weave components. In WEB, CWEB, FWEB, and SpiderWeb, file positioning also counts. Funnelweb views its source file as chunks of of code interspersed with documentation, but without restrictions on how many chunks of code, say, can follow one another without intervening text. In the others the file is deemed to made up of sections, each of which has a (possibly empty) documentation section, followed by a (possibly empty) definition section (macros) followed by a (possibly empty) code section. The only end delimiter for any one of these parts is the opening delimiter for the next part. In any case, though one may have multiple macros in the definition section, one may have at most one named code chunk in the code part. Effectively, then, the file is broken up by the @-whitespace-char or `@*' combinations that mark the beginning of the TeX part of a web section (or module, as Knuth used to call it, until remonstrations to Silvio Levy (the coauthor of CWEB) about the use of the term were made in this mailing-list).

CACM articles on literate programming

From: **Rob Beezer**
Date: 23 Jul 1992

Here are some references for some of the articles about literate programming from the Communications of the ACM. The last three articles contain some examples of literate programs not written by Knuth.

Jon Bentley, Don Knuth Literate programming Communications of the ACM, 29, no. 5 (1986) 364-369

Jon Bentley, Don Knuth, Doug McIlroy A literate program Communications of the ACM, 29, no. 6 (1986) 471-483

Jon Bentley, David Gries Abstract data types Communications of the ACM, 30, no. 4 (1987) 284-290

Christopher J. Van Wyk Printing common words Communications of the ACM, 30, no. 7 (1987) 593-599

Christopher J. Van Wyk Processing transactions Communications of the ACM, 30, no. 12 (1987) 1000-1010

From: **Cameron Smith**
Date: 24 Jul 1992

Chris Van Wyk's Literate Programming column in CACM appeared four more times in addition to the ones cited yesterday by Rob Beezer:

December 1988, pp. 1376-1385 (the last time a complete program appeared in this column)

June 1989, pp. 740-754 (fragments of a "literate" file differencing program were published, with a critique by Harold Thimbleby, the implementor of the *first* "cweb" system, which was based on Troff rather than TeX)

September 1989, pp. 1051-1055 (an article by Norman Ramsey discussing the implementation of Spider Web, the first attempt at a programming- language-independent version of Web; no program or critique appeared)

March 1990, pages 361 and 365 (not inclusive; the article was divided) (this is just a brief -- and rather sullen, I thought -- note from Chris Van Wyk stating that the column was being dropped because the only people who seemed to be writing literate programs were people who had implemented literate programming systems)

IMHO Chris Van Wyk didn't do a very good job with the literate programming column; a full year elapsed between the second and third installments, and only once was the gap less than six months. He should either have worked harder or quit sooner. Moreover, his own contributions were limited to behind-the-scenes coordination and a few paragraphs per column -- he left nearly all the writing to the guest writers and critics, and I saw no indication that in three years he had ever tried Web (or any other literate programming system) himself even once. I don't think he was entitled to ascribe the failure of the column to a general lack of interest in literate programming.

As long as we're building a literate programming bibliography (if that's what we're doing), Thimbleby's article on Troff-based "cweb" ought to be mentioned; it appeared in the Computer Journal in June 1986 (volume 29 number 3, I think), on pages 201-211.

This group is now, what, a week old? Do we need a FAQ list yet? 1/2 :-)

From: **Eric van Ammers**
Date: 23 Aug 1992

The LITPROG discussion list is a good initiative. It was very sad when Van Wyk decided to stop the literate programming column in CACM. He observed that all contributions to his column came from people who had written their own literate programming system and this made him wonder how widespread literate programming would ever become. However, he promised to continue the column as soon as people start writing literate programs using tools made by others. No doubt the LITPROG discussion list can be quite valuable in this respect.

Therefore I propose that the list of Frequently Asked Questions (FAQ) on literate programming maintains an overview of tools for literate programming, together with their particular strong and weak points. In parallel we should compile a list of desirable properties users feel a literate programming system should possess in general. I expect that from this information rather soon an literate programming tool will evolve that suits a wide audience.

A "Tempest in a Teapot"?

From: **Cameron Smith**
Date: 31 Jul 1992

Is the sudden silence on this group just a result of the TUG meeting being held now, or was the storm of interest in literate programming merely a tempest in a teapot?

I am new to CWEB and to literate programming generally, and I would be very interested to hear from people who've used literate methods to develop substantial software, especially if several people were cooperating on a project. Did literate programming make your programs better, easier to write, easier to debug, easier to maintain, and/or more efficient? Did it introduce any problems that you wouldn't have had if you had used traditional methods? A full-blown case study isn't necessary; anecdotes would be of interest.

Equally of interest would be stories from people who tried literate programming and didn't like it (although maybe such people aren't on this list!). Anybody who tried WEB or CWEB or c-no-web or some such and then abandoned it, would you care to say why? Anyone want to vouchsafe a few paragraphs for the enlightenment of others?

From: **Marcus Speh**
Date: 01 Aug 1992

Cameron, you said first: Is the sudden silence on this group just a result of the TUG meeting being held now, or was the storm of interest in literate programming merely a tempest in a teapot?

I guess it hard in general to keep both novices and veterans together on the same mailing list. For some reason, my "anecdotes" take the form of propositions [that is so because in my recent work, it is hard to be really exact...]:

PROPOSITION-1: In literate programming, novices either drop out before they have really started, or they become veterans quickly.

PROPOSITION-2: Veteran literate programmers are a special brand: used to be made fun of, not appreciated as the perfectionists who many of them are, they learnt to live alone and without a mailing list for years. Now, it is hard for them to learn to live as friends...

PROPOSITION-3: Programmers come in two different brands: literate programmers and non-literate programmers [some mouth flamed the latter as "illiterates" earlier on this list - but to justify the propositional, and therefore 'neutral' character of this note, let's be nice]. Many do not know, to which side they belong - the bright or the dark one.

PROPOSITION-4: Many programming people do not know that they actually perform literal programming. Definitely you can be a literal programmer without knowing of applying any of the kinds of WEB...life just may become harder for you.

PROPOSITION-5: Non-literal programmers cannot be easily convinced that they might gain something changing side. Literate programmers are often considered perfectionists, and for them literal programming is L'art-pour-L'art, luxury, in other words.

PROPOSITION-6: Literal programming is a luxury. But writing programs in an 'i-just-let-it-grow'-style is a luxury, too. Anything which stimulates me, wakes up my intellectual curiosity, is a luxury - but a necessary one.

I'll rather drop the line now...I am definitely to be put into the "novice" slot of literate programming - but I have thought about it a bit - probably too much compared to the work which has come out of it yet - and probably too little to satisfy your needs, Big Brother.

I'm interested to hear whether this literate programming really is a useful tool or just something for academics [...] who have the luxury of taking all the time they want on any project they want to play around with.

It is a useful tool for me. It definitely increased my level of reflection upon what I was doing. It saves me time because the programs mostly do run in the first place - it costs me time because I now like to treat many otherwise neglectable pieces of code like little diamonds - and cannot be sure that this will pay beyond the fun. It definitely costs time because I am trying to convince my colleagues that they should try *WEB, too. But I am a born missionary anyway and so this meets my needs as well.

I find it important to note though, that nobody would carry out "research" knowing in advance what will be useful...I even made the experience that scientific creativity [sorry for using otherwise obscenely maltreated words in this context] directly depended on me doing the useless. But this is not a mailing list for Hegelian dialectics, I understand. I am just trying to say that the fast way is not always the best. Being a semi-novice, that is the situation I am in [sort of] - and NOW: rise, you veterans!

From: **Timothy Murphy**
Date: 01 Aug 1992

Psst. DEK isn't listening, is he? I wonder has he got a mole in this group?

The truth is, only pointy-heads and weirdos use plain TeX nowadays. Everybody else in the world, including Dan Quayle, is using LaTeX. That's the main reason, IMHO, for the relative failure of WEB (as opposed to TeX itself). Life is too short to study webmac, when one could be listening to Enya.

From: **Marcus Speh**
Date: 31 Jul 1992

Is there some congenial relationship between DEK and Big Brother?

The truth is, only pointy-heads and weirdos use plain TeX nowadays. Everybody else in the world, including Dan Quayle, is using

LaTeX.

I agree with that. Though, I wonder..who's that potatoe Dan Quayle - did he write a Macro that I should know? I do not agree with Murphy's Guess:

That's the main reason, IMHO, for the relative failure of WEB (as opposed to TeX itself).

I am using LaTeX with (F)WEB. Never really learnt anything else but LaTeX. I don't consider WEB a failure just because it isn't a discipline by its own in Barcelona, yet.

Success relative to TeX's: ask one of the older chaps what they thought, say, 7 years ago, seeing the TeX source to what should become their paper... in 1990, still, I had to fight hard to convince anyone to use LaTeX instead of Plain TeX (plus some self-made clumsy macros to set the title page)- no, not on Galapagos, but in a major European [aha!] theoretical physics institute with lots of smarties around.

I do not think anyone could have foreseen the enormous success of TeX in the scientific world [the only mouse hole I know] - Why give up hope for literate programming yet? It is not greater computational complexity which makes people shrink back from literate programming. Unlike literate programming, TeX did not really have any serious competitor, though, I think. The success of TeX seems to be rooted in the fact that it was free from beginning and grew into an institution pretty soon - inheriting some seriousness from DEK.

Question to anyone of the more veteran literate programmers: how "institutionalized" is literate programming today? Are there meetings, special journals? Servers with special software? Experiences with trying to publish a literate program "as-is"? If not: what would you like to see? [I myself do not read any CS related journal, being a truly illiterate literate programmer.]

What can we do? For now, we can only increase the popularity of literate programming by creating a stimulating environment on this list. And sharing code perhaps. In addition to what has already been down-loaded. In case there is nobody at your place to ask: I invite anyone who has in mind to start using FWEB, or has started it not too long time ago to ask, or discuss it with me. I was happy enough to have a knowledgeable person here when I started, and that helped a lot.

From: **Kees van der Laan**
Date: 01 Aug 1992

I started with LaTeX. I am now using TeX to my staisfaction. Because it is more orthogonal, more compact and stable. Of course I don't refrain from using any TeX especially when publishing houses support the author, by providing guidelines et cetera. TUG itself allows users/authors to submit LaTeX or TeX copy, so not everybody is using LaTeX.

With respect to literate programming the idea to write for humans is not new, to supply tools which more or less force a programmer to document what he is doing is something. But as far as I see it, and at the moment not much experience with WEB, is the relational structure as opposed to the hierarchical structure, the real break through. History in database programs demonstrated the usefulness of that approach. I hope to contribute more in future to this list or to the literature of literate programming by `programming pearls.'

CWEB history

From: **George Greenwade**
Date: 06 Aug 1992

I am still admitting a huge ignorance of virtually all high-level languages, but I think it's worth noting that Knuth's name is now attached to Silvio Levy's CWEB (now Levy-Knuth CWEB). It is relatively clear that Knuth not only has suggested that better interfaces should be built -- as I understand it, he has already made enhancements and extensions to other conceptual applications of WEB in other non-Pascal languages.

From: **Timothy Murphy**
Date: 04 Jun 1992

First, I don't see the advantage of C over Pascal. Pascal supports dynamic allocation, it has an ISO standard, and it is strongly typed, which C is not. I think C is just great --- no language flame wars, please --- but I don't believe that there is one best language in which to write NTS. As well, I have spent enough time trying to port abominable C code from UNIX to DOS to have been disabused forever of the idea that C is automatically or easily portable. WEB could presumably be extended rather simply to support dynamic allocation.

I'm pretty sure that if Knuth were starting again he would use C rather than Pascal. We had the pleasure and honour of a long visit by him earlier this year, and he spent a great deal of his time hacking cweb. (This seemed the only part of TeX he was concerned with, at that time anyway.) He spoke strongly in favour of literate programming in C.

He was absurdly modest about TeX, and complained that as he went round the world he found mathematicians everywhere crouched over their terminals typing in TeX when they should be in the library! When asked why he hadn't tried to make money out of TeX, he replied that he _had_ patented 1 algorithm (I wonder which one?) He was very interested and supportive of attempts to design an Irish font, and asked to see examples of old Irish printing. His own lectures, incidentally, were all handwritten in several colours on transparencies.

From: **Daniel Kustrin**
Date: 13 Aug 1992

Just a small query. When was CWEB designed? I keep finding 1987 as the year when Silvio Levy rewrote the WEB system into C. Is this correct, 'cos I just found a University of York paper written by Harold Thimbleby entitled "Experiences of 'Literate Programming' using CWEB [a variant of Knuth's WEB]" which was written in 1984. This is a part of his introduction: "... After hearing Donald Knuth extol his literate programming system, WEB [Knuth, 1982, 1984], I decided to implement a Unix version of it, which came to be called cweb. ...". The paper is dated August 31, 1984. And it's reference number is: YCS.74(1984)

Interestingly enough in the section 8.1 he points out that "... in its current form CWEB needs a table which specifies the programming language lexemes, quoting mechanisms, and comment conventions. These features can be expressed using regular expressions... Language independence of a more limited form is often required..." He then expands on how regular expressions can be used to write a "spidery"-like system. He also looks at formatter independent system.

From: **Chris Flatters**
Date: 13 Aug 1992

Daniel Kustrin writes: When was CWEB designed? I keep finding 1987 as the year when Silvio Levy rewrote the WEB system into C. Is this correct, 'cos I just found a University of York paper written by Harold Thimbleby entitled "Experiences of 'Literate Programming' using CWEB [a variant of Knuth's WEB]" which was written in 1984. This is a part of his introduction: "... After hearing Donald Knuth extol his literate programming system, WEB [Knuth, 1982, 1984], I decided to implement a Unix version of it, which came to be called cweb. ..." The paper is dated August 31, 1984. And it's reference number is: YCS.74(1984)

There is no connection between the two CWEBs. Levy's CWEB is the CWEB that is in common use today. Thimbleby's was notable in using the standard UNIX typesetting utility, troff, rather than TeX.

From: **bbeeton**
Date: 28 Aug 1992

Recently, I forwarded to don knuth a message about CWEB, and received the attached reply. I think it's of general interest to this group. (silvio is silvio levy.) As background, I must explain that for several years, I've been acting as a semi-official agent, forwarding bug reports about tex and friends; ordinarily Knuth prefers to receive such reports on paper, but he allows me to send them electronically to his secretary. anyone who has a well-documented report of a real bug can forward it through me, or send it by regular post to knuth at the address shown in the tug membership list. bug reports sent to me for forwarding will be vetted before they go to knuth; anything that is found to be spurious, or in the nature of a suggestion rather than a bug will be politely returned to the sender.

You asked about the status of CWEB: Yes, it has indeed taken on a life of its own. I expect it will be my programming language for the rest of my life, and I'm actively maintaining it (with Silvio), currently trying to make it better for systems programming (since it will greatly improve all existing systems!!) and more portable to varieties of C compilers. The CWEB master sources are now in their own directory at labrea; the previous CWEB subdirectory of the tex sources now contains just a pointer to the main CWEB directory. WEB, on the other hand, is part of TeX and no longer being maintained except for catastrophic errors; maybe even catastrophic errors will be regarded as features, in fact, because I think WEB has evolved to a desirable stable state. Those who wish to change it have made their change files; any changes to WEB itself will screw up those numerous change files.

From: **Don Grodecki**
Date: 7 Dec 1992

I am just learning a bit about web, and I would like to know if anyone knows about a version for C++ ? Also, rather than using TeX I think that we would prefer to write in FrameMaker, and then extract a file for processing by tangle. I am pretty good at extracting such things from FrameMaker MIF files. If anyone has done anything like this please let me know. Thanks!

From: **Cameron Smith**
Date: 8 Dec 1992

I am just learning a bit about WEB, and I would like to know if anyone knows about a version for C++ ? [...]

Hans-Hermann Bode some time ago made available his patches for CWEB to support C++. Levy and Knuth are presently working on incorporating those patches, and some other new features, into a new version of CWEB, which is now being beta-tested. I expect that in not too many weeks a new official release will be available that supports C++ and ANSI C.

From: **Silvio Levy**
Date: 8 Dec 1992

Hans-Hermann Bode wrote modifications to CWEB that allow it to cope with C++ syntax, and I have incorporated these changes, with his permission, onto a beta version of CWEB. When Knuth returns to the US later this month we plan to make that the official version. But if you would like to test it out, I can send it to you.

CWEB formatting

From: **Mike Yoder**
Date: 07 Aug 1992

Daniel Kustrin wrote: Has anyone had good/bad experiences with large programs in CWEB and if so are there things that we (novices) should know.

There is a problem I encountered with literate programming which can be very serious in large programs and fairly minor in small ones. It can be described as the problem of incrementality; it can either be fixed in your literate programming tool or, possibly, in your make files and source control.

Suppose you have a large program written using literate programming techniques, and now say that you want to edit its documentation without changing anything in the code. You do so, and now your generated Ada or C or whatever looks out of date with respect to the literate programming source. So, the next time you do a "make" command, the literate programming program will run, generating a new program (which happens to be the same as the old one); its binaries now look out of date, so the compiler is invoked to recompile the source; any modules that depend on this one may also end up getting recompiled, and finally your linker will link the whole mess. The result can be a very long and time-consuming null operation.

I solved this problem by having my literate programming tool compare all generated sources to their old versions (when such existed) and to simply delete the new source if it was the same as the old. This makes all the dates look the way you want, so a "make" will rebuild just what it would rebuild if you hadn't used literate programming techniques.

Using this technique makes literate programming be an improvement over the technique of using special comments, because in the latter method make will still recompile your program if you edit only parts of the program that are inside comments. I think that this ability is essential to an literate programming system used with large programs (even with medium-sized programs, doing without it is painful). And yes, this does mean that I think the special comments method is inadequate for doing literate programming with large programs.

From: **Daniel Kustrin**
Date: 11 Aug 1992

I am having a small (suite) of problem(s) with CWEB. One of the more irritating is that it does not type set the following correctly:

```
typedef enum {
    SINIT,          /* task initialised */
    SRUN,           /* ready to run */
    STERM,          /* task terminated */
    SACCEPT,        /* waiting for corresponding entry */
    ENTRYQ,         /* queued for corresponding accept */
    /* and so on for a few more states */
} STATE;
```

Now we all see that STATE is a type. CWEB belives that it is a variable. Although @f would work I don't want to have more @f

statements then code... Also can I stop CWEB formating my C code. I am happy with the current layout of my code, and I am not happy with the way WEB attacked it and reformated it. Not nice. Can anyone help?

From: **Cameron Smith**
Date: 11 Aug 1992

Daniel Kustrin writes: [complaint about CWEB not parsing a "typedef enum" correctly...] Also can I stop CWEB formating my C code. I am happy with the current layout of my code, and I am not happy with the way WEB attacked it and reformated it. Not nice.

This is my own main complaint with CWEB: it is at once too dumb and too clever in its formatting. The idea of being able to write little "scraps" of code in any order and poof! have ctangle magically assemble them into a coherent program is really nice, but I'm still not convinced that the convenience of this is worth putting up with the manhandling that my code suffers at the hands of cweave.

After all, let's face it: until things reach the point where we can directly edit the beautifully typeset output of cweave and TeX, the major benefit of all this typographic niceness is in producing legible documentation for archival purposes and for the benefit of future programmers (including ourselves) who come in to work on our code. While we're still in the heat of creation, we look at the WEB source 10 times as often as the typeset listing. (This is my own experience; others should of course feel free to relate their own!) And that means that to get work done, we still must manually maintain a fairly sanely indented source file. To do this we each develop a layout style that makes sense to us. I for one find it most irritating to have my code reformatted into someone else's style. When the reformatting actually introduces semantic confusion because the "clever" tool has misunderstood my code, it's downright infuriating. Perhaps this is petty, but it's so. Now, if it didn't involve actually rewriting the cweave program I'd consider fixing this, doing a little customizing to suit my taste, but I just don't have the time to tinker with someone else's massive and complicated monolithic program.

Don't tell me that just tinkering with "prod.w" will fix this; it won't. Far too much of the typesetting "smarts" is hard-coded into cweave itself. For example, I wanted to change the way indenting is handled in short if/else constructs: instead of having

```
if (a>b) x = y - 2;
else t = 4;
```

I wanted

```
if (a>b)
   x = y - 2;
else
   t = 4;
```

Never mind whether this is better or worse; I wanted it. "OK," I said, "I'll just find the TeX macro that cweave emits after an if condition, and redefine it to make a line break and indentation." Not that hard to do by looking at the TeX file created by cweave and the macros from cwebmac.tex. Great. But the same TeX macro is used to do six other kinds of indentation (such as laying out type declarations of variables), so there's no way to change the if's without changing those too. Levy and Knuth just knew the "right" way to handle these things, so they "optimized" the code by collapsing two logically distinct operations into one, so I would have to re-write God knows how much of cweave to get this fixed. And of course it's not worth it, so I either grit my teeth and live with it, or I don't use CWEB.

If CWEB is really going to fly, it needs to be redesigned so that a great deal of the formatting decision-making is decoupled from the compiled code. Ideally, it would be table-driven; CWEB would be distributed with a default set of formatting tables implementing the Levy/Knuth style, but there would be a fairly transparent configuration program that each user could run to generate his/her own private layout tables. Parser generator tools like lex and yacc, or tools for generating tables from precedence grammars, already exist and could perhaps be adapted to this purpose. Also, the TeX code would use separate names for each logical function, several of which could by default be equated via \let to the same action, but with the cweave output using the different names in different circumstances so you could redefine the format of "if (a>b) x=2" without also reformatting "int x,a,b".

I will allow as how it's not reasonable to expect Knuth to have looked ahead and anticipated all of this when he designed the original WEB, and I know that Levy simply made the minimal set of changes to adapt WEB to C in making CWEB. I don't mean to disparage what they've done. But as Fred Brooks said in "The Mythical Man-Month": "Plan to throw one away; you will anyway." In "The Errors of TeX" Knuth confesses that he had qualms about letting the first TeX out for testing by others, because as long as he was the only user, if he found something about the language that he didn't like he could simply change the specification and rewrite the code. But he knew that he couldn't expect everyone who needed TeX to do a task that he hadn't anticipated to rewrite the program to make it work. But this is exactly the situation we face with cweave.

Now that we have some experience with WEB derivatives being used by many people other than their implementors, if we're still serious about this approach to Literate Programming, then we really need to think about designing a better literate programming tool. It must be more customizable, it must be better able to interact with make (no more re-compiling code because the comments changed!), etc. etc. I myself would love to take off a year or two and work just on this, but I'm not a tenured fellow of anything and haven't been endowed or granted, so I'll have to decline. But I'd be very willing to contribute to a group effort on an as-time-permits basis.

Is there a grad student out there looking for a thesis topic who'd be interested in spending a couple of years coordinating a distributed volunteer effort? Are there other literate programmer readers who'd be interested in working with a dozen or more people you've never met on a not-for-profit project? Am I simply dreaming?

Meanwhile, I've promised to tone down my comments about c-no-web, so I'll just quietly mention that it NEVER rearranges line breaks, and then leave all of you to draw your own conclusions... ;-)

From: **Richard Kooijman**
Date: 11 Aug 1992

I myself use noweb. It is simple and leaves all your code alone. You miss out on features likes @d and @f but the first is language specific and the latter not needed since noweb doesn't need hints on how to 'interpret' your code (it just copies it the way it is). It only understands @ to begin a chunk of documentation, <<...>>= to start or continue a named chunk of code and <<...>> to insert a chunk of code.

Furthermore it comes with some utilities (that may have their equivalents in other implementations, but I don't know those enough) like noroots to display the code chunks that have no parents in the hierarchy, nountangle to tangle code but put the docu in comments. And of course there are equivalents of tangle and weave. It's simple and fast and language independent.

From: **Bob Surtees**
Date: 11 Aug 1992

One of the things I like about CWEB is that cweave does format the code fairly consistantly so I don't have to be concerned about the different ways different people prefer to pretty their code. Perhaps there should be an execution switch and possibly some sort of template file that would let both fweave and ftangle format their output for those people who don't like the default. My problem is that I can't get people too excited about literate programming so when I'm done I have to use indent to pretty up the C sources so they can support it. Amazing!

From: **Steve**
Date: 12 Aug 1992

Cameron says: This is my own main complaint with CWEB: it is at once too dumb and too clever in its formatting...

and goes on to say (after a bit of griping about cweave):

It must be more customizable, it must be better able to interact with make (no more re-compiling code because the comments changed!), etc. etc...

I don't know exactly how customizable it is, but have you considered playing with something like Norman Ramsey's SpiderWeb? It allows you to specify the grammar and how you would like it formatted, which is a feature I like (I've already modified the way it handles keywords like "extern", and it only took me about half an hour, and thats starting from scratch). If you really wanted to rewrite the way weave treats your code, this is a good place to start, without having to rewrite the entire WEB code.

From: **Cameron Smith**
Date: 12 Aug 1992

Thanks for the pointers to SpiderWeb and noweb (both of which are creations of Norman Ramsey, I believe). I (obviously) haven't tried either one; I will make an effort to do so in the near future. From the brief descriptions it sounds like noweb is easier to get working in a hurry but SpiderWeb is closer to my own personal idea of what an literate programming tool should be (i.e. I don't mind having code reformatted for typesetting as long as I can control the nature and extent of the reformatting). At any rate, I've heard of both products before but never until now had a reason to try them.

From: **Bart Childs**
Date: 12 Aug 1992

Yes, they are both creations of Norman Ramsey. I am sympathetic to your views of what literate programming tools should be (rather I am in complete agreement) but I fear making them so complex that they will be harder to use.

To achieve what you want may require a significant extension of the `webmac.tex' that must be more or less standard with Spider. I think that it would be a little more direct than trying to change an existing weave, except possibly fweave. John may have put enough of that into it. It probably also helps that fwebmac.sty is a web created from fwebmac.web.

From: **Bart Childs**
Date: 12 Aug 1992

Cameron Smith's arguments are well stated. If you read the documentation that Krommes wrote in his FWEB (and it also does C and C++) you can see that he experienced some of the same pains that Dan and Cameron are talking about. I agree with them and have suggested many times that we need a "tailorable" literate programming system. Norman Ramsey also used the argument that some languages need to keep the programmers `line breaks and indentation' in his design of noweb.

One word of caution about `tailorable' literate programming systems. One good CS philosopher noted (approximately) "if you want to make a user interface unuseable, add functionality to it." The creation of a system to aid the "ordinary literate programmer" in maintaining or creating his/her own `preferred style' would be a monster. For example, in WEB/CWEB the @: format uses the \9 TeX control sequence (in FWEB it is @9). I have probably written more lines of WEB than most and have caused a large number of students to do so; none of us have ever defined the \9 to tailor anything.

I apparently am a bit different from Cameron in that I rely on the weave/tex output extensively. I do that in spite of having a nice 1280/1024 Xterminal on my desk and the printer being at the other end of the building. Also, I am convinced of the value of the nice formatting.

I have no desire for a WYSIWYG system because WYSIWYG is a lie! What you see on the screen is at best a poor approximation of what even a low resolution device like a laser printer gives. (I don't even have convenient access to a 400dpi NeXT printer much less a high resolution device.) I usually study the code in the weave/tex output stage and when editing I want to see the escapes into code mode (|code_mode|). Too many programmers will not use long names with the underscore and italicized words are easily confused with variables. His `10 times as often' is overstated for me, but he still makes a good point.

Although I am a tenured professor, I plan to put up with the indentation's ... until a willing graduate student comes along and is willing to do that work along with some tests that would help us understand its benefits, costs, ... Obviously, it does not infuriate me. I would use the word annoy. Incidentally, I also would not call it monolithic: weave.w, common.w, prod.w,

Cameron gave a good example of a need for tailoring:

```
if (a>b) x = y - 2;
else t = 4;
```

Although I also prefer his form, in textual matter it is quoted as common knowledge that the most readable information has 9 to 12 words per line. Counting each `word', delimiter, variable, and operator as a word, the first has 11 and the second has 4. It would not surprise me that a code beautifully typeset and using the `9 to 12' rule might be more comprehendable. The reason (IMHO) that we prefer these skinny presentations is that we have been trained to expect code in that fashion. I have trouble reading skinny Fortran codes, C codes with indentation based on 8-space tabs, C codes with aligned braces, and aligned right texts using monospace fonts.

In the paragraph where he mentions:

Levy and Knuth just knew the "right" way

Don and Silvio are both great talents but I am sure they meant WEB and CWEB to be simply a first step at making systems available for the writing of better codes.

I have not spent a lot of time studying Spider, but I thought that some of the `table-driven' part is there. It probably relys on a fairly small spidermac.tex/sty. That may be a place to start.

I recall a lot of the earlier submissions being about WEB systems too big to work on PCs. Wouldn't making these `tailorable' also cause even more problems in the same line? We need to keep such users in mind, but not as the primary design criteria. I believe that OS/2 and MAC-Sys 7 are both virtual memory systems.

Cameron's quotes of Brooks and Knuth are well chosen. Knuth has also said a number of times about how much he has learned from making codes available for the public to help him debug. Alas, we striving to do literate programming have lost him to volume 4 of ACP. I think that he would be pleased with these kinds of discussion and that he will leave the changes for us to design and implement.

The questions Cameron asked in his last paragraph are good and the project(s) worthwhile. I believe the bigger contributions will be in the building of better user interfaces between things like RCS, imake, dbx, and non-monolithic webs.

From: **Frank Zimmermann**
Date: 03 Sep 1992

I'm new on this list, so it could be that my question is of type COMMON. I really would like to use CWEB, but it's really suckin' that you have such a long turnaround time (WEB-> CTANGLE -> TurboC ->WEB). So my question is: Has anybody (seen) a shell for CWEB (perhaps together with TurboC) for messDos? Would be glad to hear from you...

From: **Marcus Speh**
Date: 04 Sep 1992

F. Zimmermann wrote: So my question is: Has anybody (seen) a shell for CWEB (perhaps together with TurboC) for messDos?

There was an earlier announcement by Bart Childs, concerning "web-mode". Web-mode is to be used with the GNU Emacs editor, a beautiful shell-like environment for literate programming. I have used it together with FWEB, but it knows about WEB, CWEB and FWEB. It is capable of many things, including jumping to sections and modules, inserting (and previewing) index entries, hiding and exibiting the body of a literate program file (showing the tree), inserting, quoting, and consistently renaming modules etc. It supports change files and journal files. Bart Childs says a new and much improved version will be ready very soon - I am already happy with the one I have: there are almost no flaws. Especially for large literate program files I would not like to miss it anymore.

Though it might be outdated soon, the 20 pp. User's Manual gives a nice glimpse of web-mode's features - I will send the .tex or the .ps source to anyone who asks me for it.

First contact

From: **Bryan Oakley**
Date: 18 Sep 1992

OK, I'm a newbie here. New to the mailing list, new to the concept (more or less). I have no xWEB (FWEB, CWEB, etc.) software (yet), but I do have questions.

A friend of mine has given me glimpses into FWEB, and I've looked through the past missives on this list. I must say that the correspondence is interesting. ... seems to be a mix of philosophical and practical advice. It's the practical advice I'm after. The philosophy I pretty much go along with. If I may attempt to paraphrase, literate programming is less like writing code and more like writing a technical paper, where the code is in the footnotes. Somewhat like "Here's what I intend to do (and by the way, here's how I did it)". Am I correct so far?

I have looked at an extremely limited set of FWEB code and found it at first glance to be difficult to read. I'm sure that comes with the territory; any new language is difficult at first. I was impressed, however, with the formatted documentation. My question is this: presuming that using xWEB is a better way to construct software (I assume that's the common belief...), is it wise to begin using it in the middle of a medium sized project? The software I work with is 150,00 to 200,000 lines, roughly 1500 files long. Been around since the '70s. And yes, it is in (good ol') FORTRAN. What's the practicality of using xWEB during the maintenance of said software on a routine by routine basis?

Assuming for the moment that it is practical to use xWEB on a routine by routine basis, how wise is it to use what amounts to an obscure language on a project that must be supported for many, many years to come (it's a government project...)? I think it is safe to say that very few programmers ever will learn a dialect of xWEB in college, which would make it difficult to hire new programmers without requiring additional training. It's hard enough anymore just to find programmers who know FORTRAN...

Correct me if I'm wrong, but I get the impression that most of the dialog in this group is between individuals using xWEB for mostly personal (ie: 1 person) projects. How does this fit into a larger project spaning 10 programmers or more? The biggest issue seems to be during turnover of the staff, when a new member of the development team now not only has to learn the application (and possibly hardware/software/OS of the application), but now has to learn xWEB and possibly TEX as well. On the surface I'm willing to learn xWEB myself in the interest of improving my own work, but I'm hesitant to recommend it as a primary tool for the project in general. I would like to know if anyone has tried to migrate to xWEB in the middle of a project staffed by more than one programmer.

One final question so I may appear more literate in the future, just what is the difference between the various dialects: WEB, FWEB, CWEB, c-no-web, SpiderWeb, et. al.? Is one a 'root' version with the others supersets, subsets, or both? What is the best way to reference the software in general (ie: xWEB, web, WEB, etc.)? And lastly, should I try one, which is the most robust. If it helps for that last part, my system is currently a Sun workstation, and will be some other POSIX compatible machine about a year from now.

I would also like to reinforce a suggestion made by someone else, and that is to see examples of some really good web'd code. I will gladly summarize all replies received, should they require summarizing.

From: **Allan Adler**
Date: 20 Sep 1992

I am interested in taking certain badly documented C programs and rewriting them in CWEB. I'm just learning CWEB now, partly for that purpose. I can understand how useful CWEB can be for documenting my own programs, but it seems to me that there must be special problems in documenting someone else's program, particularly if it is a program I could not possibly have written myself and

which I will have to decipher bit by bit as I CWEBify it. The program is between 500K and 1 MB of code spread out over 40 or 50 files. I have a few insights into how it works but I certainly do not understand it as a whole. Any suggestions on how to undertake such a project will be welcome. I would be glad to let someone else do it but I have been dropping hints to that effect for years without any takers, so I am pretty much resigned to doing it myself.

From: **John Yeager**
Date: 21 Sep 1992

My experience is due to trying to webify existing 370 assembler code, but may be useful. My first piece of advice is to understand the code you are documenting. While this appears a truism, it is tempting to use web to record your growing understanding of the code as you work on it. While this may work, the final product will not be particularly better than if you had merely gone in and commented the code. The fallback that seemed to work for me was to try to understand the highest level code and simulate development with structural decomposition using web. While this does not always create an optimal presentation of the code, it is usually understandable.

The other hint I can provide, is that in certain tragically spaghettied cases, the best way to add structure via web is to document the program as a finite state machine. Often that is the model a programmer using non-structured coding has in mind at least unconsciously, and can often clarify the structure of otherwise seeming random behavior.

From: **Norman Ramsey**
Date: 22 Sep 1992

In 1987 or so we started using web on a multiperson project, government funded, which is still going on. We had a big training problem. I still don't know of any satisfactory answers to that one. We were left with the feeling that using web was a win, but we weren't able to explain why in any great detail. We also came up with a long list of ways web could be improved, some of which I've addressed in a later tool, noweb. If you're interested in the details, I refer you to @article{ramsey:literate, author="Norman Ramsey and Carla Marceau", title="Literate Programming on a Team Project", keywords="web", journal="Software---Practice \& Experience", month=jul, volume=21, number=7, pages="677--683", year="1991", refereed=1, also="Princeton tech report CS-TR-302-91"}

From: **Timothy Murphy**
Date: 23 Sep 1992

What is the best way to reference the software in general (ie: xWEB, web, WEB, etc.)?

On a tiny point of information, I think "web" and "WEB" are the same thing, and refer to Knuth's original tangle/weave software, which is still an integral part of TeX software.

And lastly, should I try one, which is the most robust.

I'm no sort of expert, but CWEB has the Knuth imprimature, for what that is worth. (I feel it is basically good -- and reasonably small, which I find reassuring -- but it certainly has a number of rough corners still to be smoothed.)

From: **Joachim Schrod**
Date: 24 Sep 1992

We are using WEB systems in medium-sized projects (5-10 persons over a few years), too. Our biggest problem in training is not TeX (that must be learned by new team members anyhow since all our docs are written in TeX...). People seem to over-react on the paradigm of literate programming. Either they are totally enthusiastic or they say *NO*. They tend to forget that it's a method which helps only for programming in the small, not for programming in the large.

I usually try to show the similarities to structured programming. While one would never use structured programming to design a larger software system, it clearly has its place in the implementation of classes. IMHO the same holds for literate programming, and it combines nicely with structured programming. But that's another point of training: That program parts should be described with pre- and postconditions, that invariants are stated, etc. I can recommend @book{spec:liskov:abstraction, author = {Barbara Liskov and John V. Guttag}, title = {Abstraction and Specification in Program Development}, publisher = mit, year = 1986, isbn = {0-262-12112-3}, library = {D.1/Lis}, annote = {Well written introductionary book on abstraction entities and specification. Focuses on CLU.} }.

From: **Glyn Normington**
Date: 30 Sep 1992

I am involved in some literate programming on IBM's Customer Information Control System (CICS). Our literate programs involve a

systems-programming flavour of PL/1 rather than Pascal and IBM Bookmaster rather than TeX, but the concept is the same as WEB. The major difference is that we we don't pre-process our programs via an equivalent of WEAVE but have defined suitable Bookmaster tags to achieve pretty printing and cross-referencing. Another slight twist is that we allow formal specifications, in the Z language, to be included in literate programs so that we can reason about their correctness.

So far I have only seen Donald Knuth's "The WEB System of Structured Documentation" and a couple of articles in Communications of the ACM. I am interested in language extensions to the literate programming constructs to support multiple source files. I am considering an import and export mechanism to allow a literate program to control what program fragments it allows to be used by others (export) and what program fragments it uses from others (import) - a bit like Modula/2 modules. I am interested in any other references, text books, etc which deal with literate programming in general and literate programming language extensions in particular.

From: **Eric van Ammers**
Date: 30 Sep 1992

The first FAQ on literate programming, dedicated to FWEB, has been born. This is a good start and it will definitely help many newcomers in the field to find their way much more efficiently. However, I would like to stimulate the discussion within this group towards more general topics and some of the more recent mails suggest the time is ripe. Evidently a FAQ is to summarize each discussion.

First of all I want to support and extend the earlier made suggestion to start a discussion on literate programming tools. Not only would I like to have an overview of the specific tools that are around, but in addition I would welcome a discussion on the properties that we think such a tool should exhibit in general and which properties we consider of highest priority. By means of this discussion it maybe even possible to get to a sort of specification for a future literate programming-tool.

Secondly I'm interested in the more 'philosophical' aspects of literate programming. I'm convinced that this will be very helpful to spread the word. Although I'm a fairly experienced literate programmer myself, I have e.g. a lot of trouble explaining the superiority of the literate programming paradigm. Let me jot down a few of the questions that in my opinion belong to this category:

- What makes a literate program superior compared to more conventional techniques e.g. stepwise refined modules in combination with a (consistent) documentation?
- Is a literate program superior to a documentation that contains every refinement as a separate routine described independently? And if yes, why?
- Most people agree these days that "programming" is essentially independent of any particular "programming language". That is a "good" programmer will deliver "good" programs in any language (given a couple of weeks to learn the language), while a "bad" programmer will never produce good programs (no matter what language they use). Now, if it is true that literate programming somehow captures the essentials of "good programming", then it must necessarily be independent of any particular language. Can we define these essential characteristics of literate programming?

It seems to me that this sort of discussions are in the long run much more important and interesting then the strictly technical ones that currently dominate. A final remark about the format of FAQ's. A certain Texinfo format has been mentioned several times. But please take notice that not everyone programs literate with WEB and LaTeX. So I strongly recommend a more formatter independent format for FAQ's.

Renaming module

From: **Joachim Schrod**
Date: 23 Sep 1992

That is a problem we encountered the other day and have yet to notify John Krommes of the problem. It apparently comes about from having a module name that is an exact subset of a longer name rather than a length.

I don't know about FWEB, but in the original Pascal WEB (and in CWEB) program part names [*] must be prefix free. I think the abbrevation possibility was more easily implementable this way (when I remember correct, it's done as a binary search tree).

[*] Please, please, please: Don't use the term `module' for WEB sections. It is used in Computer Science since two decades for completely other entities. Otherwise, every serious CS person will think that the literate programming folks don't know about the profession.

Myself, I also don't speak of section names, I call them program part names. It's because they are not bound to one section, ie, they are not the name of a section. One can have sections without program parts, and named program parts may be split over several sections. (So the term `program part' is for me simply one refinement which is to be inserted somewhere.)

From: **Silvio Levy**
Date: 24 Sep 1992

I don't know about FWEB, but in the original Pascal WEB (and in CWEB) program part names [*] must be prefix free.

Yes, that is the case in theory. If you have @<foo@> you can't say @<foobar@> later, or vice versa. Actually the first case was silently allowed, which is perhaps unfortunate. In the next version of CWEB (2.8) it is planned that abbreviations of a section name can occur before the full name. To make things well-defined I then disallowed @<foo@> followed by @<foobar@>, and the actual behavior now conforms with the documentation.

[*] Please, please, please: Don't use the term `module' for WEB section.

We're slowly making the change. Version 2.8 will refer to sections, not modules. Still, there is a lot of polysemy in technical terms (and not least in Computer Science), so I think the charge that literate programmers are computer illiterate is unfounded. Version 2.8 is expected out in a week or two.

Myself, I also don't speak of section names, I call them program part names. It's because they are not bound to one section, ie, they are not the name of a section. One can have sections without program parts, and named program parts may be split over several sections.

Well, some dogs have no name, and some names apply to several dogs. Doesn't mean I'm wrong in saying that Rex is my dog's name.

From: **Joachim Schrod**
Date: 25 Sep 1992

[*] Please, please, please: Don't use the term `module' for WEB section.

We're slowly making the change. Version 2.8 will refer to sections, not modules. Still, there is a lot of polysemy in technical terms (and not least in Computer Science), so I think the charge that literate programmers are computer illiterate is unfounded.

I don't want to leave a false impression: I did not and do not think that literate programmers are computer illiterate. But when I advocated to other CS folks the LitProg paradigm in general and WEB systems in particular and gave them the docs, many complained about the usage of the term `module' (which is a software component with a specification and an implementation, the latter hopefully done in the spirit of information hiding).

I know that CWEB 2.8 is avoiding the term module now, and I applaud it. (Klaus Guntermann -- who sits in my neighbour room -- told me that you even changed all variable names like mod_... Tough. :-) :-)

From: **Silvio Levy**
Date: 26 Sep 1992

Joachim Schrod: I don't want to leave a false impression: I did not and do not think that literate programmers are computer illiterate.

I know. What I meant to say is that the claim, from whatever source, is silly. The other meaning of module is by now well-established, but that was not the case yet when Knuth introduced the name. In any case, we are in agreement about the need to change the name.

From: **Timothy Larkin**
Date: 25 Sep 1992

Joachim Schrod writes: Please, please, please: Don't use the term `module' for WEB sections. It is used in Computer Science since two decades for completely other entities. Otherwise, every serious CS person will think that the literate programming folks don't know about the profession.

I note that the CWEB documentation, which enjoys the nihil obstat of DEK himself, explicitly uses the term "module" to refer to the unit of WEB consisting of the TeX part, the definition part, and the implementation part. Regardless of its use elsewhere in CS, any "serious CS person" who would conclude that DEK "doesn't know about the profession" doesn't know about the profession.

From: **Joachim Schrod**

Date: 26 Sep 1992

Please, please, please: Don't use the term `module' for WEB sections. It is used in Computer Science since two decades for completely other entities. Otherwise, every serious CS person will think that the literate programming folks don't know about the profession.

Tim Larkins wrote: I note that the CWEB documentation, which enjoys the nihil obstat of DEK himself, explicitly uses the term "module" to refer to the unit of WEB consisting of the TeX part, the definition part, and the implementation part. Regardless of its use elsewhere in CS, any "serious CS person" who would conclude that DEK "doesn't know about the profession" doesn't know about the profession.

I know very well how much DEK has done for CS. (It's quite normal if you have an M.S. with honors in this field...) His one of the very few persons where articles of the '60s still are very influential today. He laid the theoretical foundation of many areas. Students typically read him not enough, eg, his paper about the early development of programming languages (pre-FORTRAN), his contribution to the Structured Programming debate, his seminal papers to attributed grammars should be a must.

But -- DEK is no God, he has his fields and his interests. And while I have read a very large amount of papers and books from him, I have yet to find something about software engineering. Something which addresses the difference between programming in the small vs. programming in the large.[*] His contributions (and Literate Programming is IMHO not the least among them) belong all to the former area.

This is also very good recognizable if you look at his software. It's of a typical style: monolithic, elegant, often beautiful, and not connected to the outside world, often difficult to use within other software systems. This shall in no case debase his software -- it's fitting to its requirements. But my requirements are usually different; I'm a True Believer(tm) of the need for reusable and maintainable software. LitProg helps for the latter, but not for the former. Griese's answer to DEK[**] addresses this issue better than it's possible in an email, so I don't want to repeat it here.

[*] The last time I used these terms in a c.t.t posting, I received questions. So: @article{se:deremer:large-vs-small, author = {Frank DeRemer and Hans H. Kron}, title = {Programming-in-the-Large Versus Programming-in-the-Small}, journal = ieeese, volume = {SE-2}, number = 2, month = jun, year = 1976, pages = {80-86} }

[**] @article{litprog:gries:adt, author = {Jon Bentley and David Gries}, title = {Programming Pearls --- Abstract Data Types}, journal = CACM, volume = 30, number = 4, month = apr, year = 1987, pages = {284-290} }

This paper was an answer to @article{litprog:knuth:hashtrie, author = {Jon Bentley and Donald E. Knuth and Doug McIlroy}, title = {Programming Pearls --- A Literate Program}, journal = CACM, volume = 29, number = 6, month = jun, year = 1986, pages = 471-483, }

Refer also to (just a few of the canonical publications on this theme): @incollection{se:dennis:modularity, author = {J. B. Dennis}, title = {Modularity}, editor = {Friedrich L. Bauer}, booktitle = {Advanced Course on Software Engineering}, publisher = springer, year = 1973, isbn = {0-387-06185-1}, note = {Reprinted as LNCS~30} }

@article{se:parnas:modules, author = {David L. Parnas}, title = {On the criteria to be used in decomposing systems into modules}, journal = CACM, volume = 15, year = 1972, pages = {1053-1058} }

@article{prog:appelbe:encapsulation, author = {W. F. Appelbe and A. P. Ravn}, title = {Encapsulation constructs in systems programming languages}, journal = toplas, volume = 6, year = 1984, pages = {129-158} }

@book{oops:booch:ood, author = {Grady Booch}, title = {Object Oriented Design with Applications}, publisher = benjamin, year = 1991, isbn = {0-8053-0091-0} } gives a more modern view.

From: **Daniel Kustrin**
Date: 26 Sep 1992

I know very well how much DEK has done for CS. (It's quite normal if you have an M.S. with honors in this field...)

Even if you don't have an "M.S. with honors", if you are in CompSci you will. Real programmers read Knuth in any form. Have you read Surreal Numbers or Axioms and hulls, just to name two nice works?

But -- DEK is no God...

Ah, how about alt.religion.knuth or comp.religion? I don't know about you but I think that he might be \ldots

Joking aside... you are right, he hasn't written anything about software engineering, but literate programming is a step towards it. He

integrated one of more important parts of SE with the code: documentation. OK, it's only a part of the system design and implementation in the life-cycle but it's an important step as you must know otherwise you wouldn't be reading this list. One objection I have with his approach is that is very difficult to use it with exploratory programming and it doesn't cope well with software evolution principles (although change files are a novel and interesting approach). Was there any discussion about literate programming on comp.software-eng?

How did tangle and weave get their names?

From: **Daniel Luecking**
Date: 09 Dec 1992

I know very little about WEB (actually not terribly much about programming in general). I follow this list because of an overdeveloped curiosity gland. Now I'm curious why "tangle" and "weave" were given those names. I realize that Knuth likes to play with words and the connection with the word "web" is unmistakeable, but why choose "tangle" for the process that produces the program source and "weave" for the process that produces the TeX file for documentation? (Rather than, say, the other way around.) I have something of a bad memory for names and I need hooks to hang them on. Presently I reason: Tangle and TeX both start with "T" so naturally they are not associated, thus weave produces the TeX file. I'd prefer a less perverse hook.

From: **Timothy Larkin**
Date: 09 Dec 1992

From the WEB user manual: The TANGLE program is so named because it takes a given web and moves the modules from their web structure into the order required by Pascal; the advantage of programming in WEB is that the algorithms can be expressed in "untangled" form, with each module explained separately. The WEAVE program is so named because it takes a given web and intertwines the TeX and Pascal portions contained in each module, then it knits the whole fabric into a structured document. (Get it? Wow.) Perhaps there is some deep connection here with the fact that the German word for "weave" is "web", and the corresponding Latin impaerative is "texe"!

From: **Charlie Farnum**
Date: 09 Dec 1992

Tangle produces ugly, tangled up Pascal code (tangle intentionally produces code that is difficult to read).

Weave produces a tex file that eventually produces a beautiful, tapestry-like document.

From: **Marcus Speh**
Date: 09 Dec 1992

Now I'm curious why "tangle" and "weave" were given those names. I realize that Knuth likes to play with words and the connection with the word "web" is unmistakeable, ...

Is it really? -- In his "FWEB User's Manual" [M-2.6], John Krommes is quoting DEK himself: "The name WEB itself was chosen in honor of [Knuth's] wife's mother, Wilda Ernestine Bates."

...but why choose "tangle" for the process that produces the program source and "weave" for the process that produces the TeX file for documentation?

because... "The TANGLE program is so named because it takes a given web and moves the modules from their web structure into the order required by the compilers... The WEAVE program is so named because it takes a given web and intertwines the TeX and code portions contained in each module, then it knits the whole fabric into a structured document... Perhaps there is some deep connection here with the fact that the German word for 'weave' is 'web', and the corresponding Latin imperative is 'texe'!"

From: **John Fieber**
Date: 09 Dec 1992

Tangle produces ugly, tangled up Pascal code.

If that isn't the understatement of the week... Actually, the various web systems that have evolved since the first pascal web tend not to "tangle" the code so the name may not be so appropriate any more. I'm sure it will remain for historical reasons though.

Weave produces a tex file that eventually produces a beautiful, tapestry-like document.

Tangle produces a source file that eventually produces a beautiful functioning program (in theory, practice may differ).

From: **Bart Childs**
Date: 09 Dec 1992

A quote from the original paper about literate programming (which is reprinted in the book, Literate Programming.) "I chose the name WEB partly because it was one of the few three-letter words of English that hadn't already been applied to computers. But as time went on, I've become extremely pleased with the name, because I think that a complex piece of software is, indeed, best regarded as a web that has been delicately pieced together from simple materials. We understand a complicated system by understanding its simple parts, and by understanding the simple relations between those parts and their immediate neighbors. If we express a program as a web of ideas, we can emphasize its structural properties in a natural and satisfying way."

Please don't consider this a flame (Dan Luecking's note) but the actual names are not all that important. I know that Don wrote somewhere that WEB is/was the initials of his Mother-in-Law's maiden name, Wilda Ernestine Bates. INMHO, the tenor of that statement was please don't make something out of the (non)mnemonic/(non)acronym.

I don't recall any real statement as to why tangle and weave except that they go with WEBs. You might want to think of the results of our verbs tangle and weave. The result of weaving should be visually nice. The document can be whereas the program (especially tangle output) is not.

OO Literate Programming?

From: **Paul Lyon**
Date: 22 Dec 1992

On the face of it, it would seem that the concept of literate programming is not easily adapted to the "Object Oriented" approach to programming. Let us recur to Knuth's original article on the subject (I refer to the version reprinted in Literate Programming, CLSI, Stanford, 1992, pages 99--136). There we find literate programming characterized in a way that seems to fit best with the approach Knuth took in writing TeX and Metafont. The woven web file, when printed, is to read rather like a technical article (shorter programme) or a technical book (for example, Metafont: the Program).

All of the source goes into one file, from which Weave will generate a section headings, a table of contents, a proper index for identifiers (and other things designated by the author using the Web control code), and an alphabetized listing of the code section names. The author is meant to follow a narrative order in presenting the parts of the programme; as Knuth says ``but always it is an order that makes sense on expository grounds'' (op cit, page 125). And further, "...there's no need to be hung up on the question of top-down versus bottom-up---since a programmer can now view a large program as a web, to be explored in a psychologically correct order..." (op cit, page 126). [I expect that, in most cases, there will be a number of ways of ordering that will work.]

Finally, besides the obvious benefits to the reader---either another or oneself at some remove---of a more pleasing presentation of the source and of greater clarity, another benefit claimed, one now to the author, is a shorter time required to make the code work properly. The reason for this, according to Knuth, is that in preparing a proper account of the programme while writing it, one is led to clarify one's thoughts about it; as he puts it, you go into "expository mode". In trying to write enough to explain it to someone else, you are forced to explain it to yourself! [Aside: this last, in particular, is one of the things that I most liked about Knuth's idea when I first encountered it. The more carefully I write, the better I understand what I am writing about.]

Now I have said this much about the original idea for two reasons: (1) it is an attractive idea about how to write programmes, and (2), more importantly, most of the existing Web systems are designed with this idea in mind. In particular, those features of WEB itself, and its progeny, CWEB , FWEB, SpiderWeb, and so on, that go beyond the bare bones required for tangling and weaving, are mostly about forming the index, allowing the programmer to insert things besides identifiers in the index, forming the section headings, and the like, whence my remarks above about these matters.

On the other hand, for "Object Oriented" programming, one is meant to split up a programme into a number of modules, most of which contain the interface and implementation for a single data type, or group of closely related data types, and only a few of which will contain the code that ties all of these together. Furthermore, the modules that contain the programme data types are meant to be relatively independent of the particularities of the programme one is working on. In pursuit of code reuse, one is meant to make something more akin to library code out of these, free-standing and comprehensible by itself rather than tightly integrated into one programme.

It is this last that prevents one from simply adapting CWEB or FWEB in their current versions by using the file inclusion capability. This might work for the traditional way of laying out the modules that is used, for example, in C programming, but the approach taken in C++ programming is another matter. It is instructive, I think, to consider the contrast as given by Bjarne Stroustrup in Chapter 4 ("Functions and Files") of The C++ Programming Language (Second Edition, but the same is to be found in the first edition). The example Stroustrup gives is a simple calculator programme, rather like the one the Cameron Smith dug out of Kernighan & Ritchie and used for his CWEB example (described in previous postings to this group, and available from niord.shsu.edu). Stroustrup considers how this example, might be put in modular form in the traditional C style, and then goes to do describe a different layout of the kind one would use in C++ or Ada (and Modula 2, as well, I suspect). One gets five pieces: one contains the main programme, and the other four, each with its own header file and source file, contain (1) the error handler, (2) the symbol table code, (3) the eval code for the calculator, and (4) the lexical scanning code. All this, mind you, without yet introducing classes and inheritance; it is, rather, the prelude to that. Still, the header (interface) file contains the declarations of the data structure(s), if any, together with the prototypes of the functions that provide the services that are collected in this module, and the source file contains the implementation of those functions. Add in the real apparatus of C++ and one gets a header file with the class declaration(s) and a source file with the implementation of the class "methods". The syntax and accoutrement will be different for Eiffel, or Clos, or Ada, but the style remains the same.

The obvious way to proceed is to construct a web source file from which you will generate the header file and implementation source, one that usually starts out with a description of the data type, with, perhaps, a table or itemized list giving brief descriptions of the methods (functions, procedures) for the data type, and perhaps, also one describing the fields of the data structure, and then goes on to string together the implementation of the functions, adding to each some account of its implementation where that is pertinent. In short, something that looks rather less like an exercise in literate programming as conceived by Knuth, and rather more like a UN*X man page with added source code.

Now there may be no cure for this, or it may be that I am not imaginative in the right sort of way :-) In any case there is little in the way of support in the existing Web systems for this sort of thing So far as Weave goes, I have nothing concrete to suggest at this point, rather, I hope to stimulate discussion, assuming, as I do, that the efforts to add C++ support to CWEB mean that there are others out there with similar concerns. (I have the same thing, though, on my "Web wish list" as I suspect others do as well, namely support for user configurable pretty-printing styles; my tastes in these matters being significantly different from those of the authors of CWEB).

Some things that come to mind for a C++ tangle, however, might include the following. First of all, most of us still have to contend with ``dumb'' linkers that insist on linking in the whole of a object module, even if one only uses a couple of things out of it. So it is desirable that one have a separate output file for each function or procedure defined in one of this library like modules, and further desirable that on each run of tangle, in the development phase I mean, one should generate only those output files that have changed since the last time one ran tangle. Otherwise, assuming that you already have a makefile for it least this much, you will end up recompiling the lot instead of just the one or two bits that you changed. [Funnelweb has a simple version of this; one of the reasons why I like it.] Indeed, it would be nice if tangle could generate the appropriate parts of the makefile for you (more to the point, the GNU makefile, since this will be the more easily done, I think, using GNU make).

It would also be helpful if a C++ tangle would, given the definition of a function, generate a function prototype for you if you have not already provided one. This ``feature'' will not be simple one, since tangle will have to know where to put the prototype, and that will depend on whether the function is a class method, or a friend function, or one that just happens to live in the source file in question because it is more or less related to the class(es) being defined. On the otherhand, it could be a useful one; both in saving effort, and helping to cut down on programme errors.

There are other things for the wish list as well, but I have likely said more than enough already :-) Comments, please?

From: **Lewis Perin**
Date: 22 Dec 1992

I think my experience bears on the issues raised in Paul Lyon's thoughtful posting. I've been writing C++ WEB code for several months now in CWEB (mostly) and FWEB. The rough conventions I've used seem to avoid the woes of a `monolithic' approach to C++-WEB.

The basic idea is to keep separate source files for a class's outer spec (meant to be included in any file that uses it directly) and for the class's implementation (seen only in the listing for that implementation. Say the name of the class is X; then the spec file would be xh.web, the implementation would be x.web. If class Y is derived from X, then yh.web will include xh.web. If z.web is the overall application source, it will include yh.web directly (and xh.web, of course, indirectly.) C++ adepts will by now have thought of plenty of corners of the language this simple technique fails to touch, but it does get a lot of work done while weaving output that's about as coherent as I ever get.

From: **Lee Wittenberg**
Date: 28 Dec 1992

Paul Lyon made a number of interesting points points in his message (most notably for me the one about "dumb" linkers that insist on linking in all of an object module rather than just what is needed). Since I've been doing object-oriented literate programming for a bit, I figure I ought to respond. I'm not necessarily disagreeing with Paul -- this is just how I've been doing it.

I prefer to put a single class in a source file (I believe this is what Paul said he prefers, as well), although sometimes I put related classes in a single web. I use the @(feature (available in Spidery WEB and CWEB the last time I checked) to create the header files (separate headers for each class, regardless of whether they share a source file or not). The structure of a class "Object", for example, would look like this:

```
@c
@<Header files needed for |Object|@>@;
@(Object.h@>@;
@<Methods needed for |Object|@>@;
```

Using @(Object.h@> in the code instead of a #include "Object.h" is a nice little feature of file modules: the code gets expanded both in the main program _and_ in the Object.h file. Spidery WEB complains about this (but does it anyway), but CWEB works fine (at least it did the last time I checked -- I've been using the Spidery version for Standard C and C++). I can then describe and define the Object class as I see fit, knowing that the interface will be generated automatically (and will agree with the implementation). The down side is that the header file gets updated every time the web changes, so things get recompiled that needn't be. However, there are ways around this in a makefile, and anyway, it wastes computer time rather than my time, which is much more valuable.

I've been treating the dumb linker I have to use the same way I would treat a dumb (i.e., non-optimizing) compiler. I ignore it. I figure the time I save (overall) by using literate programming more than makes up for the larger executables. Most of my libraries tend to be small, (after the third or fourth draft, anyway), so this really isn't a problem in practice. Although I will admit that if I could find a reliable intelligent linker at a reasonable price (ah, there's the rub!), I would buy it in a New York minute. Hope this adds some fuel to the fire.

From: **Joachim Schrod**
Date: 14 Feb 1993

In my mail I wanted to add a viewpoint to the (IMO interesting) question how to do indices for software systems written in an object-oriented programming language. My viewpoint is perhaps not a widespread one: I don't find identifier indices of much help. I should prepend my preconditions: In OO languages, one does not write programs any more, but modules. While designing the modules one pays attention to proper and established SE conventions, ie, one doesn't lump together specification and implementation -- these are and shall be separate documents. (Without that one can forget most abstractions.) Within a small (one-person) project, one must handle a few hundred classes, many of them from class libraries; larger systems have thousands of them.

Therefore, one does need interactive query facilities on the current state of a software system. When I want to see the declaration of an entity named by an identifier, I just click on the identifier and a window is opened where the declaration is shown. (With the restriction outlined above -- very often the declaration cannot be determined since it does not exist even at compilation time.) I can also ask where this entity is used elsewhere in this module.

An interactive access to the inheritance tree (in the current state) is also urgently needed. There one can query the method declarations which might be addressed by a given usage. (That's the software piece called a ``class browser,'' by the way.) But the definitive answer will only be available by a debugger, which must therefore be integrated in the development environment.

For me, an index is a weak substitute for the facilities of an interactive development environment, needed if I have to handle printed documents (which I try not to do at all in the context of software construction).

Oh yes, concerning printed documents: Experience shows that it is not possible to create a current printout of a non-trivial software system. This problem was described already decades ago in Brook's famous book on the Mythical Man Month. While one person is creating the document, his/her collegues change the system in their own workspaces. (Of course I assume the usage of a configuration management system.)

One shouldn't believe that one is able to create such a thing as a "current printout" if one creates a non-monolithic system. But exactly there I see the future of Literate Programming: Merging the new possibilities of Electronic Publishing (Hypertext, generic markup, connections to databases, information retrieval systems, ...) into the software construction process, and exploring new ways of software system presentation for human beings.

Arachne

From: **Jonathan Gilligan**
Date: 29 Dec 1992

Since a few folks have recently posted about wanting to program literately without using TeX as the typesetting language, let me mention that Alan Holub writes in his compiler construction book (Compiler Construction in C or something of that sort---I don't actually have the book, but looked at it while browsing in a bookstore last week) about an "arachne" system that he used to produce the code for the book. Arachne functions similarly to web, but uses C for the programming language and troff for the typesetting one. I know

nothing beyond these vague descriptions. Perhaps someone knows more.

From: **Bart Childs**
Date: 29 Dec 1992

Thimbleby wrote a CWEB using C and troff. I am sure it is available on some archives somewhere. A reference to a paper is: Harold~Thimbleby, "Experiences of `Literate Programming' using CWEB (a variant of Knuth's WEB)," The Computer Journal, vol.~29, no.~3, pp.~201--211, Jun.~1986. I believe he was at York at that time and is now at Stirling.

I never looked at it in great detail because the information I heard was that it proved that troff was extremely limiting. I suppose that the wide availability of TeX on unix boxes and the availability of Levy's CWEB also made it not nearly as desirable.

From: **Nelson Beebe**
Date: 29 Dec 1992

Jonathan Gilligan points out that Allen Holub's book uses a kind of literate programming. It has been in ftp.math.utah.edu:pub/tex/bib/litprog.* for a couple of years now; via e-mail, a message "send index from tex/bib" to tuglib@math.utah.edu will get you started.

I pointed this book out to Don Knuth, and he agreed that it qualifies as literate programming, and included it in the literature in his new book on the subject (which I've not yet obtained). BibTeX entries for these books are given below: @String{pub-PH = "Pren{\-}tice-Hall"} @Book{Holub:CDC90, author = "Allen I. Holub", title = "Compiler Design in {C}", publisher = pub-PH, year = "1990", note = pub-PH # " Software Series, Editor: Brian W. Kernighan.", ISBN = "0-13-155045-4", }

From: **Tim McGuire**
Date: 29 Dec 1992

Allen Holub's Arachne system is available from the author (holub@violet.berkeley.edu) for a fee and runs on some kind of PC system. In his words (from a longer correspondence to me in June 1990): "I wrote Arachne myself in order to write Compiler Design. It became clear as soon as I started that some sort of WEB-like preprocessor would be necessary for me to maintain my sanity, so I wrote a small one which has gradually expanded to a pretty powerful tool. I haven't written it up anywhere yet, however."

The language is C and the formatting language is troff. It is not a derivative of Thimbleby's CWEB system; it was directly inspired by Knuth's original WEB. My personal evaluation, based strictly on the Compiler Design book and correspondence with Holub, is that Arachne does fit the literate paradigm. In general, it appears that Arachne encourages larger code modules than does WEB, and thus the degree of code/documentation integration is not as high. Comments from anyone who has used Arachne would be greatly appreciated.

Literate programming, why not routines instead?

From: **Edward Keith**
Date: 04 Jan 1993

Why should I take the time to learn CWEB? What is the advantage of literate programming over extensive commenting and good design in a traditional language? I do most of my work in C. I write short functions, each with a descriptive header explaining what it does and how it does it. I comment each of the parameters and all variables when they are declared. I then run the code through a code formatter and cross reference generator. I still find the code hard to read three months later.

I read the article in the Jan. 1993 Computer Language, and have been following this list for several weeks. I find the listings in CL and posted here even harder to read than most C code (This is probably because I have been reading C for eight years, and saw CWEB for the first time last month). In the article Silvio Levy says, "The gain in clarity obtained by using CWEB over C should now be obvious." Maybe I'm a little slow. Could someone please explain it to me?

From: **Marty Leisner**
Date: 06 Jan 1993

Why should I take the time to learn CWEB? What is the advantage of literate programming over extensive commenting and good design in a traditional language? I do most of my work in C. I write short functions, each with a descriptive header explaining what it does and how it does it. I comment each of the parameters and all variables when they are declared. I then run the code through a code formatter and cross reference generator. I still find the code hard to read three months later. In the article Silvio Levy says, "The gain in clarity obtained by using CWEB over C should now be obvious." Maybe I'm a little slow. Could someone please explain it to

me?

I'm not sure about the gain in clarity. I have no problem reading quality C code months after the subject. I generally don't use a formatter. I just follow formatting rules. You really aren't supposed to look at the webbed code (from what I've seen...you look at the formatted comments, and the computer looks at the code). I too am unsure whether its worth the time to learn and whether it improves readability (although I think TeXinfo is a "good" thing -- having the documentation on line and printed).

From: **Lee Wittenberg**
Date: 07 Jan 1993

Why should I take the time to learn CWEB? What is the advantage of literate programming over extensive commenting and good design in a traditional language?

I can't speak for everyone, but there are 2 reasons why I switched to literate programming from reasonably well commented, fairly clean code.

1. Maintenance. Not only can I explain _why_ I'm doing something while I'm writing the code (these kind of explanations are not only awkward in normal programs, but often get in the way and overshadow the code), but I can also make little notes to myself about things that need to be changed, where an algorithm came from, etc (and put pointers to these notes in the index!). The index of identifiers is invaluable in trying to figure out someone else's code. In fact, my "Road to Damascus" came about when I was trying to get CWEB working on my PC. There was an obscure bug involving pointer arithmetic in CWEAVE. I spent a little over a week (during off-time at an ACM conference) poring over the woven listing, and found the bug. I would have given up on a non-literate program of similar size (and have on several).

2. Structure. Since TANGLE puts all the code sections in proper order, I can concentrate on writing the program in the order that I feel best for purposes of exposition. I can write the program for human beings, rather than for the compiler (I think this is the most important point of all). I can also modularize code without having to write a procedure (with its attendant overhead) by using a named section.\footnote{perhaps this is why Knuth used the term "module" in early versions}

In any event, I won't do without my literate programming tools. I've recently had to do some programming in PAL (Paradox Application Language). When I discovered that Spidery WEB wasn't capable of dealing with PAL (through not fault of its own -- PAL is fairly insane), I downloaded noweb, adapted it to work under DOS (a rather painful process), and now use it for all my PAL work. Even the programmers in the office who do not use noweb have no problems reading the woven listings. Hope this goes a bit toward answering your question. The best advice I can give is: "Try it, you'll like it".

From: **Mike Yoder**
Date: 08 Jan 1993

My experience with literate programming has given me a different slant on the issue than most people have; I think the principal benefit is not so much that you get good documentation (you don't always) but that you get correct documentation. I don't mean user manuals, which are generally correct but irritatingly ambiguous due to the nature of natural languages: I am referring to documents that purport to describe what a program's data structures and algorithms are.

Without literate programming, my experience says that for programs longer than, say, ten pages of code, the odds that such a document will be correct are zero. I'm not being ironic or exaggerating: I have literally never seen documentation in such cases that was useful. In most cases, either the documentation was written before the program was (and never updated), or it was written just afterward and a very incomplete job was done. I particularly remember one document describing a compiler IL that had quite correct detail about the tree portion of the IL, but was sketchy on the leaf nodes and had virtually nothing on the symbol table. It isn't all that helpful to be told that the tree node representing binary plus has two sons; you could have surmised that. But this was the easiest part of the documentation to write, and the writer was probably under heavy time pressure and trying to get as much down as possible.

It is possible, of course, to work with the program directly, but this changes maintenance from a science to an art. This is not intrinsically bad, unless human lives are involved. But in any large program where no "big picture" exists, fixing a bug consists of finding a likely-looking spot and changing it to what feels right in the hopes that it is right. If it doesn't work, you repeat the process. It would be better to know that such-and-such a routine is supposed to deal with all comments, or macro expansions, or whatever, because this can reduce the amount of code you must examine by an order of magnitude or more.

Documents get out of date because they are separated from the source, and so producing them in addition to the program becomes a two-pass process. Besides this, the documentation need not have any obvious 1-1 relationship to the program; so it may be a nontrivial task just to determine what parts of the documentation need to change after the program is modified. No wonder that most programmers take the easy way out and put off fixing the documentation forever.

Why literate programming fixes this is partly obvious and partly due to psychological effects I do not claim to understand. It should be clear, though, that when the documentation you must change is at most an editor screen or so away from your program text, there is much less of a psychological barrier to your changing it at the same time as the program. There is one other obvious reason that

literate programming helps: it makes the documentation and program be done at the same time. Once the program works, very few managers or programmers are all that keen on spending several weeks producing quality documentation; there's always other ways to spend this time that look more attractive--such as doing firefighting on the project that's behind.

From: **Eric van Ammers**
Date: 15 Jan 1993

I have been practicing literate programming for a long time and I feel very happy with it. But very often when I try to explain what literate programming is about, I get confused if people ask me what the advantages of literate programming are compared to working with small independent well documented routines. My experience teaches me that there is a big difference indeed, but until now I am not able to make this explicit. Note that Knuth in his 1984 paper in The Computer Journal also avoided this point. My question to you, LITPROG netters, is to give me your opinion and suggestions with respect to the problem above.

From: **Lee Wittenberg**
Date: 20 Jan 1993

I have been practicing literate programming for a long time and I feel very happy with it. But very often when I try to explain what literate programming is about, I get confused if people ask me what the advantages of literate programming are compared to working with small independent well documented routines. My experience teaches me that there is a big difference indeed, but until now I am not able to make this explicit. Note that Knuth in his 1984 paper in The Computer Journal also avoided this point.

The FWEB User's manual has a nice discussion of this issue (section 4.11 in the version 1.23 manual). If you can't get hold of it, I'm sure John Krommes would give permission to quote it here. Knuth does address the issue somewhat (I recall) in one of the papers in his new Literate Programming book. I'm not sure which one it was. Does anyone out there know the reference?

From: **David Kastrup**
Date: 20 Jan 1993

I have been practicing literate programming for a long time and I feel very happy with it. But very often when I try to explain what literate programming is about, I get confused if people ask me what the advantages of literate programming are compared to working with small independent well documented routines. My experience teaches me that there is a big difference indeed, but until now I am not able to make this explicit. Note that Knuth in his 1984 paper in The Computer Journal also avoided this point.

Often you cannot really avoid lengthy routines without having to formulate formal parameters, calling conventions etc. In the WEB approach (not in all literate programs, of course), documentation can include readable mathematical formulas, which I consider a boon. The problem is, that small, well documented procedures do the job as well. However, you have to formulate calling parameters and other conventions. Not only that they complicate comprehension slightly, you will simply not find any programmer intent on serious work doing that.

The advantage of literate programming over small, well documented procedures is simply psychological: splitting into sections a more complicated thing is easy and done on the fly, splitting a procedure into distinct procedures is a pain in the neck, needs additional consideration, reediting and restructuring. So it simply isn't done. Chances are, when you get both a literate programming program developed in haste and with only a small amount of documentation beside the code, and a procedural approach, that an outsider will with the literate programming understand much more after a reasonable investment of time, than with the normal program. That is because the structure of the program is more obvious, although not necessarily by being split into disjoint procedures.

From: **Mike Yoder**
Date: 20 Jan 1993

I have been practicing literate programming for a long time and I feel very happy with it. But very often when I try to explain what literate programming is about, I get confused if people ask me what the advantages of literate programming are compared to working with small independent well documented routines.

The reason you are "confused" is that the question is somewhat like being asked, "Are you still beating your wife?" The problem is the presupposition that is behind the question. There is no such thing as "small independent well documented routines" for any program of a significant size. Many people think they write them, but if you try the only empirical test that matters--namely, seeing what happens when someone else tries to use these routines when the author is gone--you will almost certainly find it works badly. Now, if you confront the author with this fact, the last thing that will happen is that he or she will say "Oh, drat. I guess they weren't well enough documented after all." What they will instead say is "that person was just too dumb to understand my code."

Please don't take this example too literally; I'm trying to get my point across in one try, and I need vivid imagery. Unfortunately, I

suspect this line of argument will be unconvincing, and you will have to find another one. There will probably be responses saying they have seen examples where my claim isn't true, but I am going to disqualify a whole slew of them right off the top. I, too, have had cases where the approach seemed to work *when the original author was available for consultation*. But this is not documentation; it is documentation plus folklore, and the folklore is usually critical. As far as I'm concerned, documentation is not adequate unless it would suffice if the original author fell under a bus and became permanently unavailable to the new programmer. This is rare, and literate programming does not guarantee it, but it makes it much more likely.

I also realize that it is possible to get by without really understanding the code; this approach "works" in roughly the same sense that Communism "worked" in the U.S.S.R. up to the point it collapsed. Good luck with your discussions.

From: **Glyn Normington**
Date: 21 Jan 1993

I have been practicing literate programming for a long time and I feel very happy with it. But very often when I try to explain what literate programming is about, I get confused if people ask me what the advantages of literate programming are compared to working with small independent well documented routines. My experience teaches me that there is a big difference indeed, but until now I am not able to make this explicit. Note that Knuth in his 1984 paper in The Computer Journal also avoided this point.

Literate programming lets you structure your program into smaller chunks without the run-time overhead of a subroutine or the effort of writing a macro. It also has the advantage that a literate program is more than a collection of program fragments as there may be high-level design documentation included which would not fit nicely into a convention program. The literate programming tools I use allow multiple programs and other files to be generated from a single literate program (which may itself be split into multiple files using an imbed mechanism). This enables better grouping of programs and data which form abstract datatypes, which our base programming language does not support.

From: **Marcus Speh**
Date: 21 Jan 1993

In my tirade, Philonous (Ph) is a friend and user of the WEB environment. He is arguing with Malevolent (Ma) who's finally going to join the literate programming family. Eventually he'll pick up a better name for himself. [Malevolent still has a hard time to believe though that CWEB++ is the "only True Web", and he will start and stay with FWEB] IMO, this socratic dialogue could actually have happened like this.

They start as suggested by Eric van Ammers:

Ma: "What are the advantages of literate programming compared to working with small independent well documented routines?"

Ph: "One can still work with small, independent routines. They're just better documented now."

Ma: "I can do well without TeX for documentation."

Malevolent obviously does not believe in DEK. You wonder who's paying him. Philonous does not really know how to answer to that. He probably does not like troff. Or maybe Malevolent has got an eye problem?

Ma: "In fact, I hate to spend to much time thinking about how to explain things to others when I haven't even finished the program."

Ph: "Before I saw WEB, I wasted lots of time finding the right balance between doc and code. With WEB, it becomes easier to write doc along with the code. And update it."

I have only experiences with FWEB, and I haven't been using it for more than one year. Before that, quite a lot of my time went into trying to improve on the delicate balance between documentation and code. None of my private efforts were really satisfying, though. Maybe also because (like many people outside of CS) I never really learnt how to program. Malevolent knows much more about programming, but he's got other things to do as well:

Ma: "But isn't this a hell of a lot of extra effort?"

Ph: "When I saw FWEB, I wasn't even put off because of the extra effort in learning something new. Though I must confess that I asked people in my field of research how long they had needed to get accustomed to the new tool."

Since then, I freely give away the magic number of "10 days"--- if you know [La]TeX and the language(s) you want to code in.

Ma: "Ok,Ok,Ok. Now, if you compare how much time it costs you and how much you gain?"

(Malevolent is a tough calculator, it seems. He obviously got the message of the zeitgeist.)

Ph: "I cannot speak for you. literate programming also is a matter of taste. It is a useful tool for me. It definitely increased my level of reflection upon what I was doing. It saves me time because the programs mostly do run in the first place - it costs me time because I

now like to treat many otherwise neglectable pieces of code like little diamonds - and cannot be sure that this will pay besides aesthetics."

Ma: "Of course I have heard about WEB. But I do not know anyone who is practicing it, really."

(Later, Philonous will tell him about the literate programming list.)

Ph: "True. The `evangelization' part sometimes is the most painful. There's no company working for WEB's success. No commercials placed. Thus, it definitely costs time because I am trying to convince my colleagues that they should try *WEB, too. But I am a born missionary anyway and so this meets my needs as well."

(He did not really have to emphasize the last point...)

The time for our key-hole listening is running out. It suffices to say that the two are having a lot more to discuss. At the end, Malevolent (overloaded with manuals, introductory texts, faqs, eager to try WEB) wants some advice how to evangelize others:

Ma: "Assuming you meet someone who's more benevolent than I am-- how're you proceeding?"

Ph: "Upon meeting someone who likewise seems to suffer likewise, and who signalizes a genuine interest in learning something new, I first show him a HUGE woven output [yes, I am carrying such a volume around mainly for that purpose]. Before putting the word "WEB" into my mouth I want to hear a SIGH when he is confronted with something which looks unlike anything he has seen yet. Even better if I have presented some more or less complicated program in a talk before: then people are lost and WEAVE's output comes handy to explain---it has got tables, it has got plots, maybe, an index, a table of contents--- Fine. Then the victim usually asks: `why did you put up all the extra work?"'

Ma: "That almost sounds like me, before I had seen the light!"

Ph: "Yes, that is the moment of truth indeed." (Timothy Murphy would much better know how to put it, I'm sure ;-)

Ph: "I start explaining some things for real [forcing the victim to recur to the beautiful output in regular intervals determined by the amount of healthy scepticism he's mobilizing to shield himself]. Eventually I show a not-too-complicated .web file. And I give him the speech which I gave you already, my friend. Of course: If I have a FWEB-FAQ (*) output at hand, I'll pass it to him, too."

You have to judge whether this may happen with your colleagues in the same way. Mine are definitely special in that many of them are used that everything comes to them pre-digested. If that is not the case, they'd rather cut on their needs: for fine documentation, for well-structured code etc. Probably this will not hold for the majority of literate programming's readers.

From: Michael Koopman
Date: 21 Jan 1993

I have been practicing literate programming for a long time and I feel very happy with it. But very often when I try to explain what literate programming is about, I get confused if people ask me what the advantages of literate programming are compared to working with small independent well documented routines. ... My question to you, LITPROG netters, is to give me your opinion and suggestions with respect to the problem above.

I am highly underqualified to respond to this question, therefore, I feel it is my responsibility to broadcast my naiveity by the widest distribution channel to which I have access, namely, the net. I admit only limited "book knowledge" of literate programming, including information from texts, journals, magazines and coffee houses. Perhaps others, like Eric van Ammers, who have first-hand, experiential knowledge of literate programming can judge the validity of the following benefits which I have presumed.

First, and foremost, literate programming provides associativity or "links" between the comments and the coding. This seems obvious in the name "Web" and a plausible influence on the name choice. This associativity knowledge is used, primitively, by literate programming compilers as I know them. That is, the links are used strictly as handles to the associated information. However, this associativity allows for "meta-compiler" activities not easily supported by well-documented code which is not literate programming. The meta-compiler activities could include such actions as automated commonality detection leading to abstraction via machine reasoning. With limited natural language processing of the comments, in conjunction with the associations identified by the code linkage, elements of the code such as contexts and intention may be derived. A meta-compiler which interprets software with such abstract knowledge makes possible software engineering methods I can not even imagine at this time. Advanced compilers could be developed which perform abstract knowledge interpretation of well-documented modules, but literate programming should make such activities easier. Qualified comments about intentional programming are requested, I merely prattle.

It also seems literate programming can help to bridge the gap between the languages. Being unlikely that one code paradigm can offer the "right choice" for all programs, literate programming should help in designing and maintaining large programs. Such are often composed of large subprogram modules in different languages. This requires an literate programming system which accepts more than one compiled language code component, e.g., C, C++, Pascal and Smalltalk.

From: Zdenek Wagner

Date: 21 Jan 1993

At the beginning I would extend the postulate about non-existence of small well documented procedures. From my own experience I know that my own small well documented procedures do not work when transferred into another program half year later. However, I can see how procedures can live together with literate programming. I am now webifying my old C++ programs. During past non-literate times I developed a bunch of general procedures and pure virtual classes which I put into private libraries in order to save compilation and linkage time. My intention for the future is to write such procedures and classes in web, compile them separately and place them into libraries. In this way I would take advantages of both literate programming and independent procedures and moreover I will save disk space since good web files tend to be long.

Defining multiple sections in one

From: **Steve Avery**
Date: 07 Jan 1993

This message is really intended just to solicit opinions on the behaviour of weave (well cweave of both CWEB and Spidery WEB - others may perform similarly though). At the moment, if I use the following code:

```
@ This is just an example.

    @<Type Definition@>=
    typedef ASCII char;

    @<Global Variable@>=
    ASCII letter;

It weaves to:

1. This is just an example.

    <Type Definition>=
    typedef ASCII char;
    <Global Variable>=ASCII letter;
```

This is not exactly how I want weave to behave, and my preferred woven output should be obvious. This type of code also tangles to the undesired output (although I must admit at least CWEB ctangle complains - I'm not sure about Spider). Now for all I know, some people using weave might take advantage of this "feature". What I would like to know is what output people prefer: the current output, or my preferred output. If people don't actually make use of this "feature", I'll probably hack around and try to "fix" it. Any comments?

From: **Bart Childs**
Date: 07 Jan 1993

I would prefer that it stop cold! I think that only one @<----@>= should be allowed per section/module.

From: **Timothy Larkin**
Date: 07 Jan 1993

I think that only one @<----@>= should be allowed per section/module.

Bart Childs in response to the observation of stevea@vast.unsw.edu.au that CWEB doesn't interpret correctly multiple @<----@>= per section.

I have found many instances in which it would be useful and logical to have multiple code parts per section. For instance, I find that a code part often consists of a function definition which tangles into @<functions@>. I would like to include a prototype at that point in the web which would tangle into @<prototypes@>. I can do this only by introducing a dummy section with an empty TeX part, which offers no advantage to the reader or the writer. Again, I may find it appropriate to introduce a global variable, destined for @<globals@>, in the same section in which I define a function, destined for @<functions@>.

Granted, such a license might be abused. But web offers as one of the primary benefits to free the programmer from the arbitrary sequence of presentation demanded by compilers, e.g. variables and functions need to be typed before use. Thus in web, unlike C, rhetorical considerations determine the order of presentation. In this spirit, it would be logical to allow additions to multiple code parts within a single section if this localizes the presentation of related elements which the C compiler requires to appear in unconnected

sections of the tangle.

From: **Lee Wittenberg**
Date: 07 Jan 1993

This message is really intended just to solicit opinions on the behaviour of weave (well cweave of both CWEB and Spidery WEB - others may perform similarly though). At the moment, if I use the following code:

```
@ This is just an example.

    @<Type Definition@>=
    typedef ASCII char;

    @<Global Variable@>=
    ASCII letter;

It weaves to:

1. This is just an example.

    <Type Definition>=
    typedef ASCII char;
    <Global Variable>=ASCII letter;
```

This is not exactly how I want weave to behave, and my preferred woven output should be obvious. This type of code also tangles to the undesired output (although I must admit at least CWEB ctangle complains - I'm not sure about Spider). Now for all I know, some people using weave might take advantage of this "feature". What I would like to know is what output people prefer: the current output, or my preferred output. If people don't actually make use of this "feature", I'll probably hack around and try to "fix" it. Any comments?

Aside from the fact that the typedef should be typedef char ASCII; it's an interesting problem. I use both Spidery WEB (which requires another @<space> before the second named section definition) and noweb (which doesn't). I have to admit that, though I like being able to define a new named section without having to remember about the @<space>, I think I prefer it the way CWEB does it. One section, one definition seems like the right way to do things.

From: **Bart Childs**
Date: 07 Jan 1993

I really don't consider it a big deal. I don't see what is wrong with

```
@ documentation
...
@<Something@>=
stuff

@ @<Something else@>=
more stuff
```

That extra `@ ' gives unique module identifiers to the index entries in `Something' and `Something else'. Tim Larkin's example is one where this is not much help. I did a quick analysis of tex.web and find that Don Knuth had 505 occurrences of "@ @<" in tex.web and these reference 356 unique module names. Most of these are quite readable and this is due to the care he took in being consistent and descriptive in the names. They are not like `Something'. Incidentally, most came from `case' statements and only four of them were `@ @<Global...' The primary reason for my thinking simpler is better is much the same as my understanding of the benefits of some of the structured programming rules/slogans/..., 1 return per function/subroutine/..., use only sequence, selection, and loop control structures. Again, it is not a big deal and simplicity and structure sure offer lots of benefits.

From: **Steve Avery**
Date: 08 Jan 1993

I seem to be getting all sorts of comments on this one, so I'll field them all here rather than in separate messages. Be forewarned, there's a bit of me standing on my soap-box here and expounding my literate programming philosophy. Feel free to shoot holes in my philosophy, but don't flame me for philosophising - you were warned. First up, Timothy Murphy <tim@maths.tcd.ie> says:

me> This is not exactly how I want weave to behave, and my preferred

me> woven output should be obvious.

tim> Not obvious to me.

Okay, I should elaborate a bit. Timothy Larkin has provided a much better example which I'll borrow (hope he doesn't mind). I'm frequently coding away, and come across a module/section which would be much more readable if I were able to include two @<section name@>= entries in it. Timothy's example would look like:

```
@*Read a String. This function....

    @<Functions@>=
    char *read_string()
    {
        .....
    }

    @<Prototypes@>=
    char *read_string();

This would weave to:

12. Read a String. This function...

<Functions>+=
char *read_string()
{
  .....
}

<Prototypes>+=
char *read_string();

Currently weave would produce:

12. Read a String. This function...

<Functions>+=
char *read_string()
{
  .....
}
<Prototypes><- char *read_string();
```

It makes a lot more sense to include the prototype in the same section as the definition. However, at present, it is not possible to do this with any of the weave's I've used (although Lee Wittenberg notes that noweb will treat it as I would like it to). The current behaviour of weave is definitely undesirable. I don't know how or why anyone would use a @<----@> section as part of an expression when there is provision for defining macros. So, I believe it should be changed one way or the other. It could either behave as I propose, or as preferred by Bart Childs <bart@cs.tamu.edu>, who says:

I would prefer that it stop cold! I think that only one @<----@>= should be allowed per section/module.

Lee Wittenberg also seems to agree with Bart's comments. Bart goes on (in a later message), to suggest the preferred solution to the problem should be:

```
@*Read a String. ...

@<Functions@>=
char *read_string()
...

@ @<Prototypes@>=
char *read_string();
```

This uses an unnamed section to provide the extra chunk of code. Bart says:

That extra `@ ' gives unique module identifiers to the index entries in `Something' and `Something else'. Tim Larkin's example is one where this is not much help.

I believe that there must be many other cases where the extra section identifier is not of that much use. Bart goes on to cite examples from tex.web by Knuth, where there are some 505 unnamed sections with no textual explanation which reference 356 unique section names. I'll go out on a limb here and say I don't much like Knuth's code. This is not so much due to the code itself, but more the

restrictions placed on it by the original WEB. The code is very flat, as WEB does not make much provision for any notion of a program hierarchy.

I code pretty much the way I was originally taught - top-down, decomposing the original problem into sub-problems, which are then further decomposed. This leads to the idea of a program hierarchy. But the original WEB only supported two levels of such an hierarchy, named and unnamed sections. This restriction in turn tends to lead to flat code, or at least code with no apparent hierarchy.

If you look at Spider of FWEB (I believe) on the other hand, they support seven levels of a hierarchy (@*=,@*,@*1-@*4,@), which leads to a more intuitive programming style. (I still find this restrictive, however, and have modified spider and CWEB to provide arbitrarily deep sectioning, but thats another story.) This idea of a hierarchy is also in keeping with the idea of the literacy of a program - how many of you would prefer a book with only paragraphs and rare chapters as opposed to everything down to paragraph marks?

Okay, next programming philosophy, I never use unnamed sections - I find doing so counterproductive. The table of contents produced by spider has the module names indented according to their level in the program hierarchy, making it easy to work out what needs to be done next, what's missing, the significance of a bit of code, etc. And now my final piece of programming philosophy, I always include some description of the code in a section. This makes maintenance easier, and enhances the overall readability of a program.

Now, given my programming philosophy, it should become apparent that using an unnamed section with no description is a big no-no for me, which is why I would prefer to be able to have multiple sections of code in a section. Anyhow, I'm probably beginning to babble, so I'll wind up here with a suggestion: either way, the current behaviour of weave should be ammended to either complain about multiple @<---@>= in a section, or to weave/tangle it as I have suggested. Votes? I'd like to hear the opinions of those who maintain WEB tools as the decision effectively remains up to them :-)

From: **Lee Wittenberg**
Date: 08 Jan 1993

Another point to add to the growing discussion: It's a lot harder for WEB to parse multiple section definitions together than to insist on separate sections separated by @*'s and @ 's. There's no way to tell (until the = is reached) whether it's looking at a section definition or use. Also, what will WEB do with:

```
x = @<Something that evaluates to an lvalue@> = 0;
or
if (@<Something interesting@> == 18)
```

or other similar examples? Both of these (slightly contrived, but similar to code I have seen) would tangle _and_ weave to something unintended if we allow multiple code definitions in a section. noweb deals with this problem by requiring section definitions to start at the beginning of a line. It's really too late to require something like that of WEB, CWEB, and the other tools that ignore spaces.

From: **Paul Lyon**
Date: 08 Jan 1993

noweb deals with this problem by requiring section definitions to start at the beginning of a line. It's really too late to require something like that of WEB, CWEB, and the other tools that ignore spaces.

A better way of dealing with this is the method adopted in Ross William's FunnelWeb programme. There are two parts to it. The first is to require that code sections be enclosed in an explicit pair of delimiters, namely an opening `@{' and a closing `@}', and the second is the requirement that any code section definition must begin with one of two directives, either `@$' (an ordinary section definition) or `@O' (a section defining an output file). Thus, a code section definition looks like this:

```
@$@<The name of the section@>==@{
..... code ....
@}
```

If the code section may be unused, one puts `@>@Z==@{' at the end instead of `@>==@{' (otherwise the programme complains about a section defined but not used, and this does count as a syntax error), and if it is to be invoked more than once, one puts `@>@M==@{' in place of `@>==@{', lest FunnelWeb refuse to tangle or weave it. Finally, if the section is to be incrementally defined one writes `+=@{' in place of `==@{', and it is deemed an error to mix the two.

None of the options apply to output file definitions; these cannot be incrementally defined, nor, sensibly enough, can one use the `@M' modifier. So it will look like this:

```
@O@<my_header.h@>==@{
#ifdef _MY_HEADER_H
#define _MY_HEADER_H 1
.... the contents ....
#endif
@}
```

Here, of course, the contents will mostly be code section invocations for the function prototypes, data structures or class definitions, and so on. Now such definitions are definitely harder to type, and Funnelweb's fussiness about the syntax is a nuisance, at least at first, but it does allow multiple section definitions following explanatory text, and thus a more free form source file, and the tighter error checking does cut down somewhat on errors in writing the source.

The major disadvantage I see is related to the one mentioned by Bart Childs, namely that allowing multiple code sections together with one TeX section complicates the indexing of identifiers, both in terms of the programming but, more importantly, in terms of the ease of finding things using the index entries. This is not an issue in FunnelWeb as it stands, because it is not language specific, so there is no provision for identifier indexing. It would, however, be a problem if one attempted to adapt the FunnelWeb scheme to a Web system with a pretty printing parser of the kind in CWEB , SpiderWeb, or FWEB.

For, we break the relationship between the numbering of the Web sections (a stretch of exposition followed by one or more code sections) and the numbering of the code sections. Now Williams dealt with this by the simple expedient of not numbering the explanatory text at all; only the code sections bear numbers, and they are only cross-referenced for definition and use with respect to each other. The organization of the exposition is entrusted to a hierarchy of section headings which get ``mil spec'' numbering ('1.2.1' for a sub-sub-heading, for example).

But if we do have an identifier (and "control text") index, and we allow, as seems proper, that an index entry can reference occurrences in the expository text (as `|identifier|' or by an explicit `@^', `@.' or `@:' directive for control text) as well as ones in the code, then we will need somehow to be able to direct the reader to the proper place. But if text passage number 33 is associated with code sections numbered 49--51, there is still apt to be confusion, even if we find a typographically satisfactory way to distinguish a text passage reference from a code section reference (say `T33' versus `C50' for one unimaginative approach that might nonetheless work). Further, I am assuming that we index things in macro definitions by reference to the text passage immediately preceeding, but I suppose we could always add a third number scheme... (Or, bite the bullet and convert to page number index entries rather than section numbers for some or all of this.)

[Aside, in FunnelWeb, there is no difference between macros and code sections: ``code section'' names can have parameter lists. The syntax is clumsy at best, and I have found little use for it. Williams made many sensible design decisions, but I would not number this one among them.] For my part, I think the complexities of the indexing scheme are not a decisive objection, but I suspect others will not agree.

From: **Steve Avery**
Date: 11 Jan 1993

Another point to add to the growing discussion: It's a lot harder for WEB to parse multiple section definitions together than to insist on separate sections separated by @'s and @ 's. There's no way to tell (until the = is reached) whether it's looking at a section definition or use. Also, what will WEB do with:* *x = @<Something that evaluates to an lvalue@> = 0;*

Okay, my two bits worth (again) - this is not a very good example of literate programming (imho of course). I was under the impression that this sort of code is best handled by the @d mechanism which web provides. I would even be quite suprised if weave was able to handle this sort of code, as I am sure it would have difficulty parsing a section which did not contain complete statements.

noweb deals with this problem by requiring section definitions to start at the beginning of a line. It's really too late to require something like that of WEB, CWEB, and the other tools that ignore spaces.

I'm sure there's a way to do it if we know that @>= ALWAYS means you are defining the code in a new section.

Paul Lyon brings up a good point (which he attributes to Bart Childs), which I must of missed some how, and that is about the problem with indexing. This is obviously a fairly big problem, seeing as without a sensible indexing scheme, maintenance of a WEB would soon become an impossibility. However, I believe it is one that may be readily solved. If it is made clear that having two code sections in one TeX section is to be an exception rather than the rule, some sort of scheme where the index numbers become 13a and 13b for section 13 could be used. This is a fairly easy index to use (although it might be a pain to generate in a one pass system), and saves the ugly T13, C24 type idea (although this would also work).

I don't like the idea of a purely page oriented scheme for the index (although it would make it easier to find the page in a hurry, it may be difficult to find the piece of code), and reckon that my proposed scheme would not separate the notion of code and text. I'm sure everyone is familiar with this sort of scheme for citations (how many people have multiple Knuth citations from the same year in their bibliographies :-)), and is fairly intuitive. Of course, if it is decided to limit one code section to ever TeX section, then the problem disappears pretty quick :-)

From: **Lee Wittenberg**
Date: 11 Jan 1993

x = @<Something that evaluates to an lvalue@> = 0;

Okay, my two bits worth (again) - this is not a very good example of literate programming (imho of course). I was under the impression

that this sort of code is best handled by the @d mechanism which web provides. I would even be quite suprised if weave was able to handle this sort of code, as I am sure it would have difficulty parsing a section which did not contain complete statements.

I agree completely with Steve that this is a bad example of literate programming and that an @d is much more appropriate. The point is that code like this exists (and there may be a more valid example out there), and the extra overhead involved in typing an extra @<space> is minimal compared to the extra overhead involved in making a more sophisticated parser that will almost certainly break existing code.

A programmer's first use of literate programming

From: **Trevor Jenkins**
Date: 22 Jan 1993

I'm five days into my first real use of literate programming. I thought that it might be of interest to some to read my observations. Firstly, some background; I'm a programmer of approximately 15 years experience. I would claim that I write good code but with the usual slips in it. Like the other day when I type + rather than - in an atoi type routine! My programming style can best be described as "middles-out" neither "top-down" nor "bottom-up" but expoliting both strategies simultaneously.

The second bit of background, I attended the London meeting where DEK gave his first presentation of literate programming. I knew this was IT. This was a methodology that I could use without changing my style or habits but which would give them some rigour and provide me with a "mechanical" aid to organise the pieces into the final program. None of my colleagues who attend that meeting saw the necessity for what we know of as WEB.

The third bit of background. I'd ban programmers from using terminals for program development! They should have to rely upon decks of cards and have to run up seven flights of stairs to submit their deck for its only run of the day. Well that's what I had to do when I started 15 years ago. However, I firmly believe that the quality of my code dropped dramatically the day I had access to a terminal. When I used a batch system I did all those things that we know we should do: desk checking, dummy runs, peer review.

Now to today. The project I'm involved in is to convert data from one proprietary format (USMARC) to another! The target format is very simple. Unpicking the rats nest of MARC format is not simple. Therefore I decided to use WEB (CWEB in reality) to write this convertor. I've so far not tried to compile the code for real and I don't have TeX readily available! I'm writting the application in the office but TeX is only installed on a PC at home. Weave and Tangle are installed at both "sites". I run the WEB source through cweave fairly frequently to check that there are no sections missing (or more likely mis-spelt); a couple of times I've run the generated C code through a compiler which has picked up a few mis-typed identifier names.

Whilst I've been working on this program I've been amazed at the ease with which the code can be written. I'd probably have written the same amount by now. I'd have tried compiling it endless times and gotten more and more frustrated. Using WEB has allowed me to concentrate upon the microscopic concerns without having to be so concerned with the macroscopic structure. Yet I was surprised by the ease with which I could flip between the sections and still hold the whole structure in my head at the same time. Even to the point of changing a major data structure mid-way, the effect that this would have had upon me in the past would only have been overshadowed by the effect upon the code itself. As it is the changes are very localised into one or two sections and the knock on effect was non-existant.

I have delayed running the woven output through TeX until tonight. The only problem with this was that I'm running CWEB 2.8 on the office VAX and 2.7 on my PC. When I saw the output I started to read and there it was (or rather there they were) errors. Sure I'd have found them eventually when I got a segmentation fault or an access violation but it was just like the early days, spot the errors when their still on the desk. Think of the journey home as the climb up those seven flights of stairs. ;-)

My boss won't apprecaite the use of CWEB. My current boss wasn't one of those who heard DEK speak indeed they don't even know how to program, they'll only see that the task is complete. It'll be seen as an unecessary complication! But I'll know that the code is solid and that it'll work properly even in the presence of incorrect data. The customer will be unaware as well.

A few things could make literate programming even better. A Windows based WEB developement editor that caught mis-typed fragment names, that detected errors in the C code (such as mis-spelt identifier names). I am glad that I went to that lecture all those years ago, I've been infatuated with WEB ever since but today I fell in love. :-)

From: **George Greenwade**
Date: 24 Jan 1993

Trevor Jenkins posted a very good overview (at least in this totally naive watcher's opinion) of first efforts at CWEB. Quite honestly, I will probably never appreciate what it is that is being accomplished by literate programming nor what is facilitated on LitProg because I am not (nor do I really desire to be) a programmer. I can read just enough code to be dangerous; indeed that's why we have the specialists in our Computer Services division as well as those of you on this list. As a long-time (18 years now; gosh, I'm a dinosaur)

non-programming user, I can appreciate his comments on limited runs, flights of stairs, cards, etc. (everyone should have to do that at least briefly to really appreciate where computing has moved in a relatively short time period). One point in the post which I would like to comment on, though:

My boss won't apprecaite the use of CWEB. My current boss wasn't one of those who heard DEK speak indeed they don't even know how to program, they'll only see that the task is complete. It'll be seen as an unecessary complication! But I'll know that the code is solid and that it'll work properly even in the presence of incorrect data. The customer will be unaware as well.

This is exactly where literate programming needs to be developed! If my understanding of what the WEB-type family of tools is correct, literate programming is very likely the most economic way of writing available to date. While the initial project may or not have been expedited, it is my guess that revisions are significantly easier since the original code fragments are intentionally and by design better documented. The customer, while initially unaware of how what got where, will be aware that at least it did. The customer will very likely need some form of support, if not another complete overhaul of proprietary-to-proprietary formats sometime in the future since that appears to be the nature of competition in proprietary software. When those times come, again assuming I am cognizant of what literate programming allows, the process should be somewhat more streamlineable. At that point, your boss will be appreciative (or at least should be).

Recognizing the managerial aspects of literate programming may very soon be a topic I pursue in the study of economics (occasionally I have to satisfy my poobahs that what I do on the net has something to do with the field of economics, so why not?). Are my views outlined above misguided? If not, I may have stumbled upon a gold mine of research topics and will willingly serve as a naive disciple attempting to win converts in the managerial side on the advantages of literate programming.

From: **Jonathan Gilligan**
Date: 24 Jan 1993

George D. Greenwade writes about the need to make managers more aware of the benefits of literate programming as a tool to enhance the efficiency of programmers. While I am a novitiate being converted to LitProg in my own programming, I fear that for people who have different concerns than mine will not agree with its benefits. (I am a scientist and I'm always most concerned that my programs produce the correct results---hence LitProg extends what I've always tried to do in my comments: to make sure that I document exactly what I think I'm doing so that years later, when I discover an anomalous result I can understand what I thought I was doing and whether the problem arises from implementing my strategy incorrectly or from using a flawed strategy. The ability to typeset equations is that much more of a blessing.)

I fear that after looking at many of the extant Literate Programs (TeX, METAFONT, bibTeX, etc.) one might conclude that to do literate programming, she must first rewrite all the systems libraries and stuff the rewritten routines into each literate program. (Why are the basic character I/O routines present in TeX, METAFONT, and bibTeX rather than stuffed into a library? This repetition of identical source code in three different programs seems like something that would make errors more likely since there is now the added problem that when one program is updated, it is out of synch with the others. Also, this kind of thing could cost more money, since now the programmer must spend more time keeeping the three programs synchronized.

The kludges that are necessary to keep common.w and common.h in CWEB synchronized (there is no mechanism to handle changes to included files and according to Silvio Levy, there are some bugs in wmerge; and there is no mechanism to have a web file output another web file containing declarations and prototypes---this feature would be a great boon to program maintenance) illustrate the difficulties that emerge in writing a small suite of related programs in the current literate style.

Clearly literate programming needs to evolve significantly if it is to achieve the status of many of the other popular methods that have come along. I think that every zealot of literate programming should read carefully the criticism of Knuth's word-frequency program from the Programming Pearls column. There is a lot of merit in that criticism and literate programming devotes should pay attention. However, thanks to Knuth and Levy (and others like Krommes and Ramsey) there is a solid foundation on which to experiment and build.

By confining my attention to some things I see as deficits to literate programming tools, I do not mean to denigrate them---there are many good tools; I just think that their merits are evident to most of the readers of this list and do not need to be repeated. Personally, I am particularly fond of CWEB even with the blemishes I have mentioned, for it is a tool of manageable complexity, so I can understand it and modify it---FWEB is just too big for me to understand in the amount of time I am willing to give it.

From: **Marcus Speh**
Date: 25 Jan 1993

Jonathan Gilligan said: Personally, I am particularly fond of CWEB even with the blemishes I have mentioned, for it is a tool of manageable complexity, so I can understand it and modify it---FWEB is just too big for me to understand in the amount of time I am willing to give it.

I agree with most of the other statements by Jonathan. I do not agree with his statement on FWEB being "just too big": FWEB is built

on CWEB. Due to excellent support by John Krommes, it has evolved into a very stable tool. If you do not insist on modifying it, there is no need to "understand" more of FWEB than of CWEB. I stick to my previous statement on this list that it won't take you more than 10 days to get used to FWEB [provided that you do not have to learn [La]TeX, your programming language of choice, nor how to use an operating system]. Moreover, for people who want to program in C++/Fortran-77 [-90]/ratfor, there is no alternative (yet). I know there are other tools who may, in principle, be adapted to any programming language, but I am thinking of someone who'd like to start right away without extensive hacking.

It is true that FWEB has a richness of structure which is apt to confuse the novice. He should keep in mind thought that there's no reason (besides sheer curiosity) why he shouldn't restrict himself to a small number of features. These will suffice to use FWEB e.g. for better bookkeeping/documentation and program-formatting and thereby satisfy a user's basic needs. While your needs grow along with your skills, you may take advantage of FWEB's superiority, since FWEB has many features built-in which are not shared by other WEB tools.

From: **Trevor Jenkins**
Date: 25 Jan 1993

My boss wont apprecaite the use of CWEB. My current boss wasn't one of those who heard DEK speak indeed they dont even know how to program, they'll only see that the task is complete. It'll be seen as an unecessary complication! But I'll know that the code is solid and that it'll work properly even in the presence of incorrect data. The customer will be unaware as well.

George Greenwade comments upon a part of my earlier posting: This is exactly where literate programming needs to be developed! If my understanding of what the WEB-type family of tools is correct, literate programming is very likely the most economic way of writing available to date.

We (on this list) are rational people, unfortunately, my boss appears not to be. The use of an argument such as the one that you suggest would be seen as specious to them! I'll use WEB anyway because I am convinced of the argument. I've tried to get them to realise that expending a little bit of effort now in order to save time in the future is well worth-while is only greeted with "we don't do that sort of thing!"

This has been a somewhat personal comment but this is the sort of up-hill fight that us Literate Programmers must contend with. On more exciting things I've got some other comments upon the use of WEB which I'll write up at the end of the week (which ought to be the end of the project) so that a fuller picture can be had. I just wish that I could provide quantative data rather than qualative assessments.

From: **Timothy Murphy**
Date: 25 Jan 1993

Why are the basic character I/O routines present in TeX, METAFONT, and bibTeX rather than stuffed into a library? This repetition of identical source code in three different programs seems like something that would make errors more likely since there is now the added problem that when one program is updated, it is out of synch with the others. Also, this kind of thing could cost more money, since now the programmer must spend more time keeeping the three programs synchronized.

Surely this was a necessary consequence of the choice of Pascal? At the time when TeX was translated from SAIL to Pascal, at least (I don't know if this is still true of official Pascal) there was no such thing as a Pascal library -- a Pascal program was a single indivisible monstrosity. Actually, WEB did meet this problem to some extent, as quite large globs of code were carried wholesale from one program to another.

But doesn't Levy's CWEB answer your point completely? His common.w (shared by ctangle and cweave) serves exactly the need you describe.

From: **Jonathan Gilligan**
Date: 26 Jan 1993

Why are the basic character I/O routines present in TeX, METAFONT, and bibTeX rather than stuffed into a library? This repetition of identical source code in three different programs seems like something that would make errors more likely since there is now the added problem that when one program is updated, it is out of synch with the others. Also, this kind of thing could cost more money, since now the programmer must spend more time keeeping the three programs synchronized.

Timothy Murphy writes: But doesn't Levy's CWEB answer your point completely? His common.w (shared by ctangle and cweave) serves exactly the need you describe.

CWEB does answer the need somewhat (although note my cavils about the difficulty of working with change files for common.h and the fact that life would be nicer if there were a tool that would let us generate common.h directly from common.w). However, my point was not that the problem I mentioned in the paragraph above was irremediable, but that a manager looking at literate programming as a possible methodology [1] might look at examples of how literate programming was used and erroneously conclude that this kind of wholesale repetition was inherent to literate programming.

I call upon the LitProg community to provide better examples of literate programming. It would be hard to improve on Knuth's skill as a programmer, but surely we can stand on his shoulders and write code in a less Proustian fashion (For a long time, Pascal used to choke on seven-bit characters...)! I conclude with a paraphrase of Strunk and White: eliminate redundant code, eliminate redundant code, eliminate redundant code. For my part, I'm trying to get used to literate programming by rewriting in CWEB a curve-fitting program I wrote in grad school. When (and if) I finish it I'll make it public for criticism.

CWEB, of course is a good step in the right direction, but literate programming as a method applied to writing tools to do literate programming is rather like meditating on one's navel (Let us remember why the literate programming column in CACM folded). We need more examples of good literate programs that accomplish real tasks (i.e., that are not simply tools to write literate programs). If many people would supply code for real literate programs, we could see a diversity of styles emerge and let these inform our own. (Kudos to Cameron Smith, BTW, for a step in the right direction for giving us a nice example (although of a toy program) of multimodule programming under CWEB).

[1] Language query: I've always hated the word "methodology," as it seems to me that it would more appropriately describe a science that studies methods ---why, then, are so many people, including myself, unable to break ourselves of the habit of using this ungainly word when we really mean "method," but don't find that simple word sufficiently pretentious? Nonetheless, the stuffiness of the word seems appropriate when discussing managers who tell their underlings how to think about writing code ;-)

From: **David Thompson**
Date: 27 Jan 1993

I was amused by Jon's ruminations about the word "methodology." FWIW, I find its overuse offensive as well. Another word in this category is "utilize." Perhaps we should agree to "use" a "method" instead of utilizing a methodology? <ducking and running>

From: **Stephen Fulling**
Date: 27 Jan 1993

Jonathan M. Gilligan writes: I call upon the LitProg community to provide better examples of literate programming. ... For my part, I'm trying to get used to literate programming by rewriting in CWEB a curve-fitting program I wrote in grad school. When (and if) I finish it I'll make it public for criticism.

CWEB, of course is a good step in the right direction, but literate programming as a method applied to writing tools to do literate programming is rather like meditating on one's navel (Let us remember why the literate programming column in CACM folded). We need more examples of good literate programs that accomplish real tasks (i.e., that are not simply tools to write literate programs). If many people would supply code for real literate programs, we could see a diversity of styles emerge and let these inform our own. (Kudos to Cameron Smith, BTW, for a step in the right direction for giving us a nice example (although of a toy program) of multimodule programming under CWEB).

After two years of reading about literate programming and thinking and talking about what a great idea it is, over the holidays I finally wrote my first literate programs. I took two short programs written 12 years ago in C and redid them in CWEB. Yes, I know that's not the way you're supposed to do it, but I wanted my first hands-on experience with CWEB syntax to be free of extraneous concerns such as correctness of algorithms.

These programs "accomplish a real task": they calculate the terms in an asymptotic expansion approximating the solution of a system of differential equations. They may therefore be of interest as examples of CWEB that are not computer utilities, such as word-counting programs. They are probably short enough (7063 and 11532 bytes) to be posted directly to the list, but I doubt that George Greenwade would appreciate that precedent, so I'll ask his advice on how to make them available.

The originals were the first C programs I ever wrote, in 1981. In 1987 I revised them and added copious comments; at that time I flattered myself that they were already fairly literate :-). But the cramped nature of program comments and especially the impossibility of including typeset mathematics left the explanatory material inadequate. The WEB versions produce 3 times as many printed pages, but the additional information is well worth it, I think. I hope that one can actually understand the purpose of the program now without reading the associated journal article in SIAM J. Math. Anal. Of course, I modularized the programs too (perhaps overdoing it a bit).

From: **Eric van Ammers**
Date: 28 Jan 1993

Promoting literate programming - I'm trying to promote literate programming, but unfortunately I live in a hostile environment. So I

need hard arguments. It would be an enormous help if I had reports on (controlled) experiments with literate programming. For example like the one of

Ramsey and Marceau

Literate Programming on a Team Project

Softw.Pract.&Exp. 21, 7, 677-683 (July 1991).

Also references to discussions on the merits of literate programming are welcome, as long as they are in refereed journals. Of course I already have the literate programming columns of CACM.

From: Trevor Jenkins
Date: 29 Jan 1993

This week's installment of "A programmer's first WEB". Nobody objected to last week's posting about writing my first Literate Program so I've taken the liberty of wasting more bandwidth by writing about this week's progress. I changed the experiment (as a result of last week's message) in that I kept notes during the week; last week's report was just Friday night recollections.

There are three sets of goals that my experiment ought to satisfy:

1 Donald E. Knuth's. As espoused in his writings on literate programming. Unfortunately, I've mislaid my copy of the Computer Journal with his original paper on the subject. :-(Must be somewhere in my loft. Also, few of the London bookstore's stocked his "Literate Programming" book and even fewer are likely to re-stock it. :-((All in all I shall refrain from saying anymore about his criteria/goals until after the project is finished.

2 My employers. As I wrote earlier this week they don't know that the experiment is being conducted; all they do know is that I'm writing the program to specification. Their goals are that the job is done with minimal effort in creating the utility and that it requires little effort to maintain it next year when the customer's data changes!

3 My own goals. I can wax lyrical about these. :-)

a I wanted to use WEB (actually CWEB) on a real project. I had read DEK's paper; the stuff in Jon Bentley's "Programming Pearls" and then Van Wyck's (short-lived) "Literate Programming" column in the Comm of the ACM but now I felt it was time for me to do it myself.

b The desire to use WEB was based upon my expectation of the complexity of the code. (Anyone who has ever tried to de-block a USMARC tape will understand what the problems are---because that is exactly what I'm doing.) I have to admit that I normally get "flustered" by the complexity of code that I've written after only a couple of hundred lines---I estimated that this conversion utility would be around 1,000 to 1,500 lines in total, ie more than I wanted to have to cope with using my old skills. So I unilaterally decided that I would write the thing using CWEB.

By the way, I've gotten the tape de-blocking working and some of the USMARC to internal format working. The CTANGLEd code for which is currently just over 500 lines. I have the reports/listings code to write and possibly some other manipulations of the data will be necessary. This latter because the specification of the data on the tape is not very clear---in fact it's opaque!

c Reduce the debugging time. I think that this has happened. The major problem is that I'm not a C programmer. Until this project I was always a Pascal programmer but the employers have switched to C so... It does mean that the utility can be run on a Unix box if required, in addition to the VMS box that it's targeted for. Even if the debugging time is not reduced at least I'm calmer than I would be when trying to get the beast to run using my old working methods! :-)

d I wanted to see what my own literate program would be like. It's rather like the first time that I held the card deck for the first production program I wrote. A great sense of satisfaction.

Summary of the week

Rather than bore you all with a diary of events I want to make some observations about my usage of literate programming. In passing I'll make suggestions for additional tools that would have made the programming task easier. The use of literate programming meant that recommencing work on Monday morning was "instantanious". The reduction of complexity inherent in each section of code meant that I could concentrate upon specific portions that required correction and concentrate upon the specifics. The index to the listing proved invaluable to getting on with it straight away.

It was easier to concentrate upon the code. Fragments of it remained in my mind longer than such things do normally. Even when I was working on fragment A, I found myself correcting fragment B and then going back to A again without loosing my train of thought. It was easy to insert new code quickly especially for a couple of things which had been forgotten. Estimating the amount of work involved was/is difficult. KDSI is inappropriate for a literate program as there are many lines of what would only be exchanged in conversations between programmers (if at all).

I made the same "slips of the pen" that I would have made in a non-literate program. However, they came to light very quickly. Usually during a code reading. Correcting complex boolean conditions became easier, though it was still difficult to express them correctly. On a number of occasions I wanted a CWEB macro feature. Not the @d directive but rather some way to have CWEB macros for

handling code that is very similar. I resorted to cut-and-paste within the editor. Any tendency to hack or patch the code could be resisted easily.

Working practices

A working pattern emerged very quickly. I commute to my office by London Underground which means that I get to sit down for approximately one hour before actually arriving at the office. I used this time to read through the listing in preparation for the days work. Reading the listing also took my mind of the delays in the journey---that is I arrived at my usual time...late. :-)

On getting to the office I would correct the errors that I'd seen during that early morning read-through. A lot of these early-bird worms were "speelnig misteaks". Once these were corrected I could start on some portion of the code that had (so to speak) taking my fancy during the journey. This thinking/editing work would continue until about 15:30 when I would try to weave/tangle the web source. Once I'd corrected typing mistakes in the .w file I could try compiling the .c file. When I had a clean compilation I then ran the executable. Whilst one iteration of the edit/compile/link/test cycle takes longer with WEB I think that the cycles were more productive than before. What fascinated me was that I rarely needed to read the source code to figure out what was the problem. Because, as I said earlier, I had it all in my head I could correct the problems quickly. (Well quicker than I used to when using the non-literate programming paradigm.)

Some "negatives"

It was a mistake to add several new sections into the .w file at the same moment that I used my first change file. I nearly "lost it" then. However, i decided that in future I would add new sections immediately prior to the Index section until such time as I wanted/needed to generate a complete listing when I would re-organise the exposition of sections in a more instructive order.

Several times I wished for a Windows-based editor. I wanted to be able to view the current section, the section in which it was used and the definition of variables and macros at the same time. Clicking on a lexeme whilst pointing at it and having all the relevant sections available would have been a great boon. I remember arguing this self-same case in an examination answer during my Master's degree finals; these finals took place not long after I heard DEK's London talk.

On my PC I have Microsoft's Programmer's Workbench which I used to compile CWEAVE and CTANGLE. I was impressed by the fact that during symbolic execution of the code it (PWB) honoured the #line directives. Shock and horror then to discover that VMS debug did not honour them! My next task is to write a simple program that takes the output from the VMS DIFFERENCES utility and creates a change file. This will of course be a literate program.

Portable C/F/WEB

From: **Philip Rubini**
Date: 22 Feb 1993

I am keen to use a WEB variant but am concerned about the question of portability of my web code. I am developing a modest program of approximately 20000 lines of Fortran which will continue to be developed in the future, hence the desire to document the code. However the actual source code will be distributed to a number of different groups (those funding the research !) - here is my concern - if I write the code in a WEB variant then I must distribute not only my code but possibly a much larger WEB environment. It seems to me that a well written web code will not resemble the destination language to any great extent hence the second problem - those I distribute the code to must learn the Web syntax if they wish to further develop the code, which they probably will.

What I think I need is a small literate programming tool that allows embedded documentation but also keeps the destination code fairly visible so that I can argue that I am using a documented form of, in my case, Fortran, that is readily understandable to someone familiar with Fortran. Obviously all of the features of a Web variant would be usefull, such as contents lists, indexes, variable lists/xrefs etc.

From: **Norman Ramsey**
Date: 22 Feb 1993

Look into David Hanson's and Robert Sedgewick's loom tool. It is not a true literate programming tool (in the Thimbleby sense) but it does enable a semblance of literate programming using standard source files and it will work with fortran. see the article in the (july?) 1987 CACM.

From: **Osman Buyukisik**
Date: 23 Feb 1993

I used FWEB a little, but I think you might want to use FunnelWeb. This WEB is language independent, and the tangled code looks good compared to the code produced from FWEB. Wowen code does not look as nice as FWEB's. This is due to the language

independence of FunnelWeb. The program does not need any extra style or tex files (FWEB does), so distribution is easier, the manual is a lot smaller than FWEB's. Smaller number of constructs need to be learned with Funnelweb. I just built mine, think it is a good literate programming tool.

From: **Lee Wittenberg**
Date: 25 Feb 1993

Philip Rubini writes: What I think I need is a small literate programming tool that allows embedded documentation but also keeps the destination code fairly visible so that I can argue that I am using a documented form of, in my case, Fortran, that is readily understandable to someone familiar with Fortran. Obviously all of the features of a Web variant would be usefull, such as contents lists, indexes, variable lists/xrefs etc.

You might want to look at Norman Ramsey's noweb (available via anonymous ftp from princeton.edu in the pub/noweb directory. It's programming-language independent, and is compatible with either plain TeX or LaTeX. It also includes a tool called NOUNTANGLE, that "transforms a literate program into a traditional commented program, without loss of information and with only a modest penalty in readability."\footnote{from Ramsey's article, "Literate-Programming Tools Need Not Be Complex", included in the noweb distribution.} NOUNTANGLE is designed so that if the literate programming "experiment is unsatisfying, it is easy to abandon".\footnote{Ibid.} noweb is mostly a collection of shell and awk scripts, so it should run on almost any UNIX system. It can also be converted (painfully) to run under MS-DOS (if you need a DOS version, don't go through the agony yourself -- I'll be glad to send it to you).

From: **Philip Rubini**
Date: 26 Feb 1993

Recently I posted the following question regarding the portability of Web programs .... I am keen to use a WEB variant but am concerned about the question of portability of my web code. I am developing a modest program of approximately 20000 lines of Fortran which will continue to be developed in the future, hence the desire to document the code. However the actual source code will be distributed to a number of different groups (those funding the research !) - here is my concern - if I write the code in a Web variant then I must distribute not only my code but possibly a much larger Web environment. It seems to me that a well written web code will not resemble the destination language to any great extent hence the second problem - those I distribute the code to must learn the Web syntax if they wish to further develop the code, which they probably will.

What I think I need is a small literate programming tool that allows embedded documentation but also keeps the destination code fairly visible so that I can argue that I am using a documented form of, in my case, Fortran, that is readily understandable to someone familiar with Fortran. Obviously all of the features of a Web variant would be usefull, such as contents lists, indexes, variable lists/xrefs etc.

The suggested possibilities are summarised below :- 1) Use FWEB for its features but suffer its complexities and pursuade those who receive my programs to learn FWEB or let them update only the Fortran code. 2) Use FunnelWeb - simpler than FWEB, produces readable program code, no pretty printing. 3) Use noweb - even simpler, no macros etc., produces readable program code. 4) Use Loom. Not a Web variant, extremely simple -suitable for documenting existing codes (?).

At present I think I can use both Loom and noweb, Loom (or a Fortran version of it) for existing programs and Noweb for new programs. Personally I found FunnelWeb, although much simpler than FWEB, to still have a confusing syntax with @'s everywhere (I would prefer readable words like 'Begin macro' etc.). I do think that pretty printed code would be nice (pretty !) - do (La)Tex pretty printers exists for C and Fortran outside of the literate programming environment? Any other comments or additions would be most welcome.

From: **Osman Buyukisik**
Date: 26 Feb 1993

I just learned of another literate programming tool that claims to be universal: CLiP from netherlands (sun01.info.wau.nl). But it is not ready to be distributed, will be by March 15. The only problem is that it works on only VAX VMS and MSDOS!

From: **Lee Wittenberg**
Date: 26 Feb 1993

Philip Rubini asks: I do think that pretty printed code would be nice (pretty !) - do (La)Tex pretty printers exists for C and Fortran outside of the literate programming environment?

There is a cprog.sty file for typesetting C code. It's available via anonymous ftp somewhere (I forget where -- anyone?). It does a pretty good job of typesetting C, and is also supposed to work for C++, Pascal, and Modula-2.

Alternatives to TeX and WEB

From: **Glyn Normington**
Date: 23 Feb 1993

Do we realy need to use TeX and WEB?

No. We use IBM's Bookmaster markup language to do literate programming. We use tags to define and reference code fragments and have an equivalent tool to Tangle which we run as a pre-processor to our compiler. We don't need to Weave since the formatting logic is built into the definition of the literate programming tags.

Why not write literate programs in a wordprocessor or some kind of WYSIWYG desktop publishing tool and extract the code from there. It seems to me that this approach would be far more programmer friendly. Such a system could make use of hypertext links to requirements, analysis, designs, and other source documents.

Although Bookmaster is not WYSIWYG, we can use a syntax-directed editor to edit Bookmaster source in a readable form.

Does any such system aready exist? The reason why I ask is because I am thinking of writing such a tool as a final year project. Any suggestions?

Our system exists but is internal to IBM. However, if you define unambiguous ways of delimiting and referencing code fragments, it is fairly simple to write a Tangle-like program to produce compilable code. I suggest you follow the usual approach and make this program self-generating which will probably require you to write a basic prototype to get you off the ground. Our prototype was a 230 line REXX program (REXX is an interpreted language a bit like PL/1).

From: **Edward Keith**
Date: 23 Feb 1993

Do we realy need to use TeX and WEB? Why not write literate programs in a wordprocessor or some kind of WYSIWYG desktop publishing tool and extract the code from there. It seems to me that this approach would be far more programmer friendly. Does any such system aready exist? The reason why I ask is because I am thinking of writting such a tool as a final year project. Any suggestions?

I think it's a great idea. I've been considering writing a suite of Winword macros with this aim myself.

From: **Osman Buyukisik**
Date: 23 Feb 1993

Stephen Cross writes "Do we really need to use TeX and WEB?" I beleive the people who talk about using wordprocessors for program development have not used any of the litprog tools. One of the biggest reasons for using the tools is not because they produce good looking code with documentation, but because they allow the programmer to use the tool as a program design language (PDL), and either top-down or bottom-up design the program without being restrained by the target language compilers. The same comment goes to noweb people, since that tool does not expand macros and tangle.

From: **Timothy Murphy**
Date: 23 Feb 1993

Why not write literate programs in a wordprocessor or some kind of WYSIWYG desktop publishing tool and extract the code from there. It seems to me that this approach would be far more programmer friendly.

I think it's a great idea. I've been considering writing a suite of Winword macros with this aim mysself.

Isn't this more or less what TANGLE is? I suppose you could add an in-built editor, but is it really worth it? You want WEAVE to output ASCII instead of TeX? Surely that can't be too difficult. Just cut out all the TeX commands it produces. (Whether that is an advance or

not is another matter ...)

From: **Preston Briggs**
Date: 23 Feb 1993

People who do work on unix don't usually have a wordprocessor handy. Usually, we have a generic text editor, a make facility, a C compiler, and (hopefully!) tex. It's certainly easier to use (or even write) a tangle and weave than it is to write a wysiwyg tool; though I think a nice tool would be a joy to use.

From: **Norman Ramsey**
Date: 23 Feb 1993

The same comment goes to noweb people, since that tool does not expand macros and tangle.

Be careful. cnoweb does not tangle, but it uses cpp to expand macros (as did CWEB the last time i looked). noweb tangles but does not expand macros. it says here that noweb doesn't have to expand macros because it works smoothly with m4 and cpp.

From: **Edward Keith**
Date: 23 Feb 1993

Why not write literate programs in a wordprocessor or some kind of WYSIWYG desktop publishing tool and extract the code from there. It seems to me that this approach would be far more programmer friendly. I think it's a great idea. I've been considering writing a suite of Winword macros with this aim mysself.

Isn't this more or less what TANGLE is? I suppose you could add an in-built editor, but is it really worth it? You want WEAVE to output ASCII instead of TeX? Surely that can't be too difficult. Just cut out all the TeX commands it produces. (Whether that is an advance or not is another matter ...)

Who said any thing about ASCII? Winword (aka Word for windows) is GUI. You can include mathematical formulas, graphs, you could even scan in your photo (as someone on this list suggested). I'm not a TeX expert, but I suspect Word for Windows can do anything it can do, with less of a learning curve.

From: **Joan Boorstein**
Date: 24 Feb 1993

What the heck is WYSIWYG?

WYSIWYG -- is an acronym (as if you needed one more) standing for 'What you See Is What You Get'. It was used (if I recall correctly) to refer to a screen editor/word-processor whose behavior allows a text's screen 'appearance' (this is not a good word choice but I can't think of a better one at the moment) to be the same (or nearly so) as its paper one.

From: **Eric van Ammers**
Date: 24 Feb 1993

It is very well possible to program literately without being tied to a particular text processing environment and/or programming language. We have been doing exectly this for a long time using the VAMP tool which recently has been replaced by the more up-to-date CLiP (Code from Literate Program). More about CLiP in a separate posting.

From: **Paul Prescod**
Date: 24 Feb 1993

What the heck is WYSIWYG?

What you see is what you get...like a graphical word processor. TeX would be more appropriately described as WYWIWYG, What you WANT is what you get, but it would fail the WYSIWYG test because the formatting codes look nothing like the final output.

From: **Eric Scharff**
Date: 24 Feb 1993

Who said any thing about ASCII? Winword (aka Word for windows) is GUI. You can include mathematical formulas, graphs, you could even scan in your photo (as someone on this list suggested). I'm not a TeX expert, but I suspect Word for Windows can do anything it can do, with less of a learning curve.

I've never used Microsoft Word for Windows, so I can't comment on its power as a desktop publisher. However, from what I know about its style sheets, if you have a good understanding of the file format, it shouldn't be difficult to WEAVE WEB output that would use these style options (for code and such.) However, are you suggesting a TANGLE that takes a Microsoft Word for Windows file as input? That sounds a bit more complex...

From: **Timothy Murphy**
Date: 24 Feb 1993

Who said any thing about ASCII? Winword (aka Word for windows) is GUI. You can include mathematical formulas, graphs, you could even scan in your photo (as someone on this list suggested). I'm not a TeX expert, but I suspect Word for Windows can do anything it can do, with less of a learning curve.

Doubt if you'll sell that one here ...

From: **Paul Lyon**
Date: 24 Feb 1993

Who said any thing about ASCII? Winword (aka Word for windows) is GUI. You can include mathematical formulas, graphs, you could even scan in your photo (as someone on this list suggested). I'm not a TeX expert, but I suspect Word for Windows can do anything it can do, with less of a learning curve.

I agree on the learning time for GUI based word processors, but TeX has several advantages here that one needs to bear in mind. First of all, TeX runs on much larger variety of platforms than any GUI based word processor at present (so far as I know, at any rate), and this seems unlikely to change any time soon.

Secondly, because TeX is (deliberately) limited to fonts that have TeX font metrics files, it is able to do a better job of formatting than word processor sytems that must cope with the vagarities of a thousand and one printers. TeX is, after all, a typesetter, not a word processor.

Thirdly, though its macro language is not easy to use, it does give one a certain flexibility in formatting that GUI based word processors do not often have. Adjustments in layout can be made as precisely as one likes by altering TeX formatting parameters, repetitive bits can be reduced to macros, and so on.

Fourthly, TeX is public domain. Not only is one not tied to a proprietary system, but one can also benefit from the contributions of others to the publically available pool of TeX (and LaTex, AMS Tex, etc.) macros and macro packages. One need only obtain a copy of the TeX-index from niord to see what is available. Provided that something suitable is out there, one may be able to get around part of the difficulties of TeX's command language courtesy the time and effort that someone else has already put in. For example, if one wants attractive tree diagrams in the woven source, one can use the TreeTeX package that Anne Brueggemann-Klien and Derick Wood put together. (Mind you, I have not tried TreeTeX with CWEB or FWEB, so I do not whether there are any name conflicts in need of resolution, or the like, that might prevent use of TreeTex as is.)

For me, at least, given that I have already invested the time and energy to learn TeX (however imperfectly :-), these considerations are sufficient.

From: **Stephan Eggermont**
Date: 25 Feb 1993

Who said any thing about ASCII? Winword (aka Word for windows) is GUI. You can include mathematical formulas, graphs, you could even scan in your photo (as someone on this list suggested). I'm not a TeX expert, but I suspect Word for Windows can do anything it can do, with less of a learning curve.

Apart from the problems noted by others, I see a few others:

- TeX has a lot less bugs, which is very important to those of us with deadlines;
- there is no way to automatically change font style for the (pascal) keywords in Word (at least the Mac version, does wordbasic help?);
- the interchange format of Word, RTF, is not very well defined (at least in the document Microsoft makes available), and is subject to changes.

Extracting code from an RTF document is rather trivial when you define a code style, just copy the part after the '{\codestyle' till the corresponding '}' into the code file.

From: **Jonathan Gilligan**
Date: 25 Feb 1993

I've never used Microsoft Word for Windows, so I can't comment on its power as a desktop publisher. However, from what I know about its style sheets, if you have a good understanding of the file format, it shouldn't be difficult to WEAVE WEB output that would use these style options (for code and such.) However, are you suggesting a TANGLE that takes a Microsoft Word for Windows file as input? That sounds a bit more complex...

Actually, WFW has a pretty powerful macro language (which bears a strong resemblance to BASIC, surprising no one who's followed Bill's excellent adventure) and it might be possible for an ambitious WFW programmer to write a weave/tangle pair in WFW BASIC, avoiding the problems of reading a weird file format, and with the added feature that a Windows hypertext help-file could be generated at the same time (a hypertext woven output with hotlinks between different parts of a module would be a fantastic alternative to reading the unformatted WEB or printing it out every day or two---I'm contemplating building a texinfo-like interface to CWEB, but I need to become much more familiar with WEB programming to be able to make appropriate design decisions, so this idea is not even at the vapor stage yet). I am not intimate with WFW, since I like TeX and don't want to spend lots of time now learning a new system, but the possibilities seem promising for the more WYSIWYG oriented.

It's possible that this could even be a quasi-real-time incremental WYSIweave, although I'm not sure how one should handle cross-references to still-undefined symbols. The great advantage this would give a programmer is that I often want to put a figure into my documentation and doing this with TeX and CWEB is enough of a pain that I avoid doing it. If I could sketch a figure in a drawing program and cut-and-paste into my WEB, that would be a real treat.

From: **bbeeton**
Date: 25 Feb 1993

I'm not a TeX expert, but I suspect Word for Windows can do anything it can do, with less of a learning curve.

Maybe it's not really relevant to literate programming, but some of us represent publishers. there are only a few text processing systems that produce publishable-quality mathematics, and word for windows is not one of them. tex is the only one that is not proprietary, and available at prices that our authors can afford.

From: **Osman Buyukisik**
Date: 25 Feb 1993

I think a lot of people are just talking about the look of the document. But how about the code fragments and macro expansions that are done by the WEB systems? I don't think that can be accomplished easily!

From: **Edward Keith**
Date: 26 Feb 1993

I've never used Microsoft Word for Windows, so I can't comment on its power as a desktop publisher. However, from what I know about its style sheets, if you have a good understanding of the file format, it shouldn't be difficult to WEAVE WEB output that would use these style options (for code and such.) However, are you suggesting a TANGLE that takes a Microsoft Word for Windows file as input? That sounds a bit more complex...

That's what I'm suggesting. I'm not sure how to do it.

Who said any thing about ASCII? Winword (aka Word for windows) is GUI. You can include mathematical formulas, graphs, you could even scan in your photo (as someone on this list suggested). I'm not a TeX expert, but I suspect Word for Windows can do anything it can do, with less of a learning curve.

Doubt if you'll sell that one here ...

I realize that this violates the religious dogma, and that no rational argument can have any effect. So I will not press it. "There is no got but DEK, and TeX is his holy word."

From: **Tony Coates**

Date: 26 Feb 1993

I think a lot of people are just talking about the look of the document. But how about the code fragments and macro expansions that are done by the WEB systems? I dont think that can be accomplished easily!

I suspect that by writing a separate program under Windows, and hot-linking to the WFW file via DDE, that you could do the macro expansions, etc., if it seemed infeasible via WordBasic. The code for this could be taken largely from current WEB-type code, with the additions necessary for handling DDE. It should be possible, as with other DDE hot-links, to then have the macros updated whenever the sources are updated. At least, in theory this all should work anyway, but I'm not saying that I've yet done enough Windows programming to be sure.

From: **Stephen Cross**
Date: 26 Feb 1993

In my previous post to this group I wrote: Do we realy need to use TeX and WEB? Why not write literate programs in a wordprocessor or some kind of WYSIWYG desktop publishing tool and extract the code from there. It seems to me that this approach would be far more programmer friendly. Such a system could make use of hypertext links to requirements, analysis, designs, and other source documents. Does any such system aready exist? The reason why I ask is because I am thinking of writting such a tool as a final year project. Any suggestions?

Well all the followup posts have been very helpful and my current idea for a final year project is to: (1) Look at extending the method for object oriented literate programming. (2) Look at how analysis and designs can be incorporated into a literate document. (3) Write Literate programs direct into a WYSIWYG word processor (ie write the woven document, no need to weave it from a source file). (4) Look at how hypertext links could be used to improve the method. (5) Write a tool to extract (Tangle) the code from the word processor. This could be done from within the word processor using macros, or a command line tool which processes a document file. (6) The tool/method will support C++.

I have not decided what operating system or word processor to use yet but it is likely to be one of Unix/Framemaker, Windows/Framemaker, Windows/Word, Windows/Wordperfect, or Windows/Amipro. It all depends on what I can get access to. Framemaker or Word seem the most likely. Your comments are welcome.

From: **Lee Wittenberg**
Date: 26 Feb 1993

I think a lot of people are just talking about the look of the document. But how about the code fragments and macro expansions that are done by the WEB systems? I dont think that can be accomplished easily!

True. But I suspect that it can be done. I suspect that a MS Word-based WEB would provide a TANGLE menu item, which would produce the C, Pascal, or whatever code (maybe even call the compiler!). The tangling algorithm could be "borrowed" from any of the extant tangle tools, and if internal macros are desired, Kernighan & Plauger's "Software Tools" books provide a lovely little macro preprocessor that could (probably) be translated into WordBasic.

From: **Timothy Murphy**
Date: 26 Feb 1993

Who said any thing about ASCII? Winword (aka Word for windows) is GUI. You can include mathematical formulas, graphs, you could even scan in your photo (as someone on this list suggested). I'm not a TeX expert, but I suspect Word for Windows can do anything it can do, with less of a learning curve. I realize that this violates the religious dogma, and that no rational argument can have any effect. So I will not press it. "There is no got but DEK, and TeX is his holy word."

What I doubt if you will sell is that Word for Windows will do anything TeX will do. Especially printing mathematical formulae, which you explicitly cite. Do you claim that it does this as well as TeX? If so, could you point to some published mathematical work to support your claim?

From: **David Kastrup**
Date: 26 Feb 1993

Who said any thing about ASCII? Winword (aka Word for windows) is GUI. You can include mathematical formulas, graphs, you could even scan in your photo (as someone on this list suggested). I'm not a TeX expert, but I suspect Word for Windows can do anything it can do, with less of a learning curve. I realize that this violates the religious dogma, and that no rational argument can have any effect. So I will not press it. "There is no got but DEK, and TeX is his holy word."

Come on, take a break here. TeX works with ASCII, or even stranger character sets. TeX is free, and available for almost any platform. Even if you disregard that Winword must be paid for, and paid for dearly, you cannot presume that everybody will only work under PCs, and don't presume that DOS is the only thing (apart from its succubus Windows) which will ever be of interest on these gadgets. Besides, different versions of that word processing package have a difficult time understanding each other. You can rely on TeX remaining absolutely compatible, not because it is the best thing possible around, but because it is a reasonable good and versatile thing, intended to stay fixed and free. And you get complete source as well for the literate tools. That ensures you that you can do a change of computing platform painlessly. Winword would fix you on one platform, and one system typo, and one OS, and payable as well. While this might be acceptable to a certain degree in companies, it is certainly not so in an academic field, if you want to do something of general interest.

From: **Trevor Jenkins**
Date: 26 Feb 1993

I think a lot of people are just talking about the look of the document. But how about the code fragments and macro expansions that are done by the WEB systems? I dont think that can be accomplished easily!

True. But I suspect that it can be done. I suspect that a MS Word-based WEB would provide a TANGLE menu item, which would produce the C, Pascal, or whatever code (maybe even call the compiler!). The tangling algorithm could be ``borrowed'' from any of the extant tangle tools, and if internal macros are desired, Kernighan & Plauger's "Software Tools" books provide a lovely little macro preprocessor that could (probably) be translated into WordBasic.

I have long dreamt of such a tool. Shortly after hearing DEK's London talk I wrote in answer to a question in my Master's examination that with (the then emerging) powerful workstations ever programmer should have a WEB system available. As a result of my recent real use of literate programming (reported last month) my thoughts have returned to the features that such a tool would provide. My programming environment of choice (given that I paid for it with my hard-earned pounds) is MS-Windows the discussion of using WordBasic is interesting, but I think misguided.

As a programmer I need assistance during the writing of the code. I need to be able to examine another section of the (potentionally incomplete) WEB. I want to be able to scroll through the code section, the macros and the text description for any one section of the text. I want to be able to point at a ``variable'' and have the details of the item shown to be, irrespective of whether it is a named code section, a macro, a variable or something that I've forgotten to define/declare. I don't think that WordBasic excellent as it might be (and I intend to invest more of my hard-earned cash upon a copy soon) will actually be up to performing all those things.

From: **Peter Olsen**
Date: 27 Feb 1993

Let me suggest another approach to Stephen Cross and his project... I don't think that the objection most people have to WEB, CWEB, FWEB, and friends is the "back end" output of TeX (which looks wonderful and is a pleasure to the eye), but rather the front end (which often looks like the set of characters you get when your parity is wrong). I'm a regular TeX user, and I still find all the special WEB constructs intimindating.

One way around this is to try to fix only that which is "broken" --- the front end. I don't think that "full WYSISYG" is the best way to do that. I think that most WYSIWYG word processors spend too much time on their screen presentation and too little on making it easy to write. Even then their screen presentations aren't true WYSIWIG because the paper never looks exactly like the screen. I think that there's a broad gulf between raw WEB and FrameMaker or Microsoft WORD and that there are some tools that will let you nicely fill it.

One is, for example, Lucid Emacs 19 running under a windowing system. As I understand it (unfortunately, my installation isn't working very well yet), you can change the color or font of individual words to mark different sections of text. These changes could indicate all the different attributes of WEB tokens. Because emacs already contains a powerful programming language (emacs LISP) built right in, it ought to be straight-forward to write out the high-lighted file with the equivalent WEB construct replacing all the colored highlights. Now just run weave and tangle on the output (for the "back end"). In fact, in many windowing systems it ought to be possible to recognize when a file is "WEB-able" and pop up a window showing the current page as it would be output using an already existing previewer. This seems to me both to be a much more feasible project that writing a new WEB on top of either Framemaker or Microsoft WORD and, as Lucid Emacs is under the GNU Public License, it seems more in keeping with the noncommercial tradition of TeX.

From: **Edward Keith**
Date: 27 Feb 1993

What I doubt if you will sell is that Word for Windows will do anything TeX will do. Especially printing mathematical formulae, which you explicitly cite. Do you claim that it does this as well as TeX? If so, could you point to some published mathematical work to support your claim?

I have never used TeX for mathematical formulae. I have used Word for Windows. It is very easy, and I have never had any complaints about the results. However, I have never published anything. A friend of mine used TeX for her thesis (very math intensive). She now uses Word for Windows, and says she will never go back.

Well all the followup posts have been very helpful and my current idea for a final year project is to: (1) Look at extending the method for object oriented literate programming. (2) Look at how analysis and designs can be incorporated into a literate document. (3) Write Literate programs direct into a WYSIWYG word processor (ie write the woven document, no need to weave it from a source file). (4) Look at how hypertext links could be used to improve the method. (5) Write a tool to extract (Tangle) the code from the word processor. This could be done from within the word processor using macros, or a command line tool which processes a document file. (6) The tool/method will support C++.

I have not decided what operating system or word processor to use yet but it is likely to be one of Unix/Framemaker, Windows/Framemaker, Windows/Word, Windows/Wordperfect, or Windows/Amipro. It all depends on what I can get access to. Framemaker or Word seem the most likely. Your comments are welcome.

Sounds like an excellent, and very useful project to me. Keep us posted.

From: **David Kastrup**
Date: 28 Feb 1993

I have not decided what operating system or word processor to use yet but it is likely to be one of Unix/Framemaker, Windows/Framemaker, Windows/Word, Windows/Wordperfect, or Windows/Amipro. It all depends on what I can get access to. Framemaker or Word seem the most likely.

That is one of the things going to make it very hard for you convincing anyone here. As you said for yourself, you have to decide what commercial OS you use, and what commercial Word Processor. So your tool would be available for only a limited circle of users, and only on an architecture probably seeming archane in a few years. I have developed serious programs and libraries under CP/M once. Most in the bin by now. The TeX/Weave/Tangle approach is very future-safe, because it works on a vast variety of systems, and is freely available. It works, for example, admiringly well on linux.

Second problem, most of these people on this list already have worked with TeX. Although starters are difficult at first, the result achievable are remarkable, and once you have invested the initial learning time, you can produce amazing results in amazing time. And when you can use TeX, the rest of the Weave/Tangle/Web design is a breeze to master. Honestly, I would rather not want to invest any time needed in WYSIWYG systems to do the markup of typeset texts myself. I am glad having TeX to do that for me.

From: **Tony Coates**
Date: 02 Mar 1993

In response to some of the suggestions that Word for Windows is a lesser medium for doing literate programming than is a combination like Emacs/TeX, I have to say that while I myself am an Emacs/LaTeX user, because of my need to typeset a lot of mathematical expressions, and my preference for the formatting that I get with LaTeX, I have many colleagues who far prefer Word for Windows because they find it easier to use, and like to see how it looks at the moment that it is written.

The point is, if one compares the number of users of a program like WordPerfect or Word for Windows to the number of users for (La)TeX, I suspect that the former are firmly in the majority. In the same way, if literate programming is ever going to hit the mainstream, I suspect it will only be on the back of a commercial package which has some sort of WYSIWYG/hypertext/ browser front-end (throw in whatever other buzz-words are necessary) which combines the functions of current programming environments with the additions necessary to actually create literate programs.

Sure, I prefer myself not to use proprietary programs, and will probably stay with the Emacs/(La)TeX brigade, but it's only one solution, and I suspect not the one that will prove to be the most popular in the long run. There has to be room for both approaches; if not, literate programming might just fade away as yet good idea that never caught the imagination of the world at large. Anyway, another 2c worth.

From: **Lee Wittenberg**
Date: 02 Mar 1993

For what it's worth, I agree with Tony Coates' observations that the WinWord/WordPerfect/etc. crowd probably greatly outnumbers the La/TeX crowd and that if literate programming is ever going to go "mainstream" it will need to be available in a WYSIWYG form. I'm also with Tony in that I'm not likely to forsake my La/TeX tools for proprietary commercial ones. My 2d worth as well (why is `d' the symbol for `pence'?).

From: **Eric van Ammers**
Date: 02 Mar 1993

The point is, if one compares the number of users of a program like WordPerfect or Word for Windows to the number of users for (La)TeX, I suspect that the former are firmly in the majority. In the same way, if literate programming is ever going to hit the mainstream, I suspect it will only be on the back of a commercial package which has some sort of WYSIWYG/hypertext/ browser front-end (throw in whatever other buzz-words are necessary) which combines the functions of current programming environments with the additions necessary to actually create literate programs.

I support Tony's vision. But it seems to me that it will be pretty laborious to design and build an literate programming tool for every combination of programming language and text processing environment. Exactly for that reason I suggested some time ago to that we should work towards an literate programming tool that is independent of programming language and text processing environement...

From: **Paul Prescod**
Date: 03 Mar 1993

The point is, if one compares the number of users of a program like WordPerfect or Word for Windows to the number of users for (La)TeX, I suspect that the former are firmly in the majority. In the same way, if literate programming is ever going to hit the mainstream, I suspect it will only be on the back of a commercial package which has some sort of WYSIWYG/hypertext/ browser front-end (throw in whatever other buzz-words are necessary) which combines the functions of current programming environments with the additions necessary to actually create literate programs.

Sure, I prefer myself not to use proprietary programs, and will probably stay with the Emacs/(La)TeX brigade, but it's only one solution, and I suspect not the one that will prove to be the most popular in the long run. There has to be room for both approaches; if not, literate programming might ust fade away as yet good idea that never caught the imagination of the world at large. Anyway, another 2c worth.

Perhaps we should be encouraging someone to make a Word4Windows-type program that produces TeX code. Perhaps someone could even write a document filter for Word4Windows to do it. The way document filters are implemented in W4Win is very extensible (they are dynamically loaded).

From: **Tony Coates**
Date: 03 Mar 1993

The point is, if one compares the number of users of a program like WordPerfect or Word for Windows to the number of users for (La)TeX, I suspect that the former are firmly in the majority. In the same way, if literate programming is ever going to hit the mainstream, I suspect it will only be on the back of a commercial package which has some sort of WYSIWYG/hypertext/ browser front-end (throw in whatever other buzz-words are necessary) which combines the functions of current programming environments with the additions necessary to actually create literate programs.

I support Tony's vision. But it seems to me that it will be pretty laborious to design and build an literate programming tool for every combination of programming language and text processing environment. Exactly for that reason I suggested some time ago to that we should work towards an literate programming tool that is independent of programming language and text processing environement...

Put it this way - I'm not suggesting that literate programming tools will be built for every editor/programming language combination. Quite the opposite, I'm suggesting that only the most popular combinations will win. I don't expect that most users care whether an literate programming tool is general or not - how many users use more that one word processor and one or maybe two languages? Most users don't need this, and so in my opinion the winning company in the future is unlikely to provide such generality. Oh, there will be some tools around that have this, for the small group of users who need it, but I can't see this really being a mainstream need.

Literate programming and Ada

From: **Frank Pappas**
Date: 24 Feb 1993

Eric van Ammers writes: Who can help me on references describing the c-no-web and the AdaWEB system?

This post is somewhat lengthy because in addition to answering part of Eric's answer, I want to take the opportunity to let this group know about what I'm doing with Ada and literate programming. AdaWEB is an early effort to provide an Ada-based WEB written. It was developed by Y. C. Wu and T. Baker at the University of Florida. Adatangle differs from other tangle processors in that it

produces formatted source code. Adaweave bunches statements like the original weave processor and doesn't support include files nor file modules.

I am writing a new Ada-based WEB, called AWEB, that properly formats the entire Ada language, supports include files, file modules, and much more. A short overview appears at the end of this posting. I'm just putting the finishing touches on Aweave, the weave processor, and hope to have Atangle finished in about two months. AWEB will be released with the same type of copyright that CWEB has and will be placed on the appropriate ftp and mail server sites. I intend to provide two versions of Atangle. One to produce unformatted Ada source and one to produce formatted Ada source. Wu and Baker gave me permission to enhance their Adatangle and use a CWEB-like copyright, so I will use that as starting point for the formatting tangle, which I will call fangle for now.

A formatting tangle is useful for people who need to provide formatted Ada source for contractual obligations but who prefer to write in WEB. For that reason fangle will copy Ada comments to the source file and not produce section number comments. I'm interested in hearing what people think should be done with the TeX part of a section. Should it be copied as inline comments? (In the example that follows -- indicates an Ada comment. Ada is not case sensitive but I use uppercase for predefined Ada identifiers.) Consider the following sections:

```
     @ Finally, here is the body for the |prompts| package.

     @<Body of |prompts| package@>=
     with TEXT_IO; use TEXT_IO;
     package body prompts is
       @<|display| body@>
       @<|confirmed| body@>
     end prompts;

     @ This procedure is provided to allow the author to annoy users
       by asking for confirmation on even the most trivial operations.
       This is decidedly user-unfriendly.

     @<|confirmed| body@>=
     function confirmed(s : string) return BOOLEAN is
       c: CHARACTER;
     begin
       display(s);
       loop
         GET(c);
         @<Return if the user responded correctly@>
         display(s, bad => TRUE); -- tell user response was invalid
       end loop;
     end confirmed;

     @ As another example of
     user unfriendliness, we make the user respond in upper case.

     @<Return if the user responded correctly@>=
     if c = 'Y' then
        return TRUE;
     elsif c = 'N' then
        return FALSE;
     end if;
```

should fangle produce:

```
     -- Finally, here is the body for the |prompts| package.

     with Text_IO; use Text_IO;
     package body prompts is

       ... omitted body of display

       -- This procedure is provided to allow the author to annoy users
       -- by asking for confirmation on even the most trivial operations.
       -- This is decidedly user-unfriendly.

       function confirmed(s : string) return BOOLEAN is
         c: CHARACTER;
       begin
         display(s);
```

```
      loop
        get(c);
        -- As another example of
        -- user unfriendliness, we make the user respond in upper case.
        if c = 'Y' then
          return TRUE;
        elsif c = 'Y' then
          return FALSE;
        end if;
        display(s, bad => TRUE); -- tell user response was invalid
      end loop;
    end confirmed;

  end prompts;
```

Should it only copy Ada comments or should it be a user option? What about code enclosed within | |, \&{}, @t...@>? What about TeX control sequences?

Overview of AWEB

```
  - Formats the language as illustrated in the Ada reference manual;
    in particular:
      *  all the variations of the select statement
      *  choices in variants, case statements, and exception handlers
      *  nested blocks
      *  subprograms with long parameter lists:
           if the following appears in the AWEB file, it formats to
           a single line if it fits:
             procedure name(arg_1: type_1; arg_2: type_2;...arg_n: type_n);
           otherwise, it formats to:
              procedure name
                          (arg_1: type_1;
                           arg_2: type_2;
                           ...
                           arg_n: type_n);

   - Include files as in CWEB, with a flavor of FWEB:
       @i and @I controlled with separate switches; might use @i
       for boilerplate like Mil-Std 2167A, and @I for program
       documentation.
   - Web macros with parameters as in spider
   - Style files and command line arguments
       * ALL COMMAND LINE SWITCHES can be placed in a style file
       * nested and default style files
       * separate switch for size of each main data structures:
         max_bytes, max_texts, max_modules, max_names, max_scraps,
         max_tokens, max_refs, stack_size, input line length,
         output line length;
         for example max_refs can be specified as -mb10k or -mb10240
       * separate switches to control special treatment of assignment
         operation, relational operators, and/or operators, negation,
         and membership operations.
   - Limbo option for placing user limbo text in generated file
   - TeX and LaTeX support selectable from command line switch
        Features that depend on LaTeX output routine may not work;
        I try to do something about this for LaTeX 3.0
   - Special formatting of Ada predefined identifiers:
        identifiers, such as CHARACTER, TEXT_IO, and TRUE, are
        supplied via a text file; user can altered file contents to
        reflect particular compiler\environment.
   - Special formatting of user predefined identifiers:
        like Ada-predefined, but formatted using different macro;
        can be used to highlight identifiers from project lexicon, like
        motor, volt, cell, CPU, etc.
   - Switch to allow automatic xref of single letter identifiers
   - Automatic xref of operator symbols used in infix form
   - Index entry for identifier declaration is underlined; entry
     for body is in italics.
```

```
- Special formatting of based numbers:
     #16#EF9# formatted as $EF9_{16}$
- Atangle will probably allow command line replacement of web macros
- Atangle will allow command line selection of which file modules
  should/should not be generated. (Eventually I plan to optionally
  generate only those modules that have changed, but I won't promise
  this for the coming release)
- custom identifier as in CWEB
- custom_i identifier:
    similar to custom, but identifier is placed in index, not the
    control sequence, and if the identifier is followed by a
    parenthesized list, the list is treated as the parameter list for
    the custom control sequence:
    * example:
          % place this in limbo
          \def\Inty(#1,#2){\int_{#1}^{#2}f(x)dx}

          -- place the following in a definition section
          @f Inty TeXi

          -- place the following in a code section
          s := Inty(a,b);

      Then Inty, not \Inty, is placed in the index and
          \|s := \Inty(\|a,\|b);
      is generated in the code section.
```

In addition to Aweave and Atangle, all the examples in the Ada reference manual in have been rewritten in AWEB. I also developed some torture tests. Once I release AWEB I will add any user reported problems to the test suite. I am writing all of my commercial software in AWEB so I intend to make whatever corrections or enhancements are necessary to keep it production quality. (Also, note the name of the company.)

I know there some literate programming tools available, but since my network access is limited to e-mail, ftpmail, and mail servers, I'd appreciate any explicit pointers to these tools. I will modify these to support AWEB and add them to the distribution. When AWEB is ready I will post a short notice to this group. As I said before, that should be in about two months.

From: **Joachim Schrod**
Date: 25 Feb 1993

Frank Pappas wrote: AdaWEB is an early effort to provide an Ada-based WEB written. It was developed by Y. C. Wu and T. Baker at the University of Florida.

Is this system freely distributable? May you point us to a location where to get it from?

I am writing a new Ada-based WEB, called AWEB

Hmm, there is already an AWEB, written by U. Schweigert. I'm posting retrieval info RSN.

From: **Frank Pappas**
Date: 26 Feb 1993

AdaWEB is an early effort to provide an Ada-based WEB written. It was developed by Y. C. Wu and T. Baker at the University of Florida.

Joachim Schrod writes: Is this system freely distributable? May you point us to a location where to get it from?

First, let me correct a mistake on my part. Wu and Baker are from Florida State University, not the University of Florida. As for distribution, I have a copy but I don't think it would be proper for me to distribute it. I will check with Baker if he still wants to distribute it. However, you should be aware that AdaWeave is written in Pascal and AdaTangle is written in Ada. Both are Sun dialects.

I didn't mention it in my earlier posting, but my Ada web is written in portable Ada and should compile and execute on a PC AT clone. I have 286 and 386 Ada compilers from two different vendors, so I will make sure it works properly under DOS and hopefully Microsoft Windows. I will distribute executables for DOS. By the end of the year GNU Ada will available for DOS and (I think) UNIX, so you won't have to go out and buy an Ada compiler.

There is a change in plan about releasing my AWEB. Someone is seriously considering using it on a significant Ada project. The project, which I won't mention unless they decide to go ahead with using AWEB, will be a substantial example of literate programming that doesn't involve TeX or web. The project will have a great deal of visibility in the Ada programming community and in the U.S.

Department of Defense, and will be highly visible to the general programming community as well.

Because I think this project provides an excellent opportunity to demonstrate the applicability of literate programming, because the project leader likes literate programming, and because I'm a really nice guy, I have promised him that I would have the complete weave and the nonformatting version of tangle ready in about two weeks. Some of the items I planned for tangle that are not essential, like command line replacement of macros, will be postponed. Anyway, when I this release ready I will make it a general release and place it on the appropriate archives. The formatting version of tangle and the features I don't put in now, should still be ready in about two months.

I am writing a new Ada-based WEB, called AWEB

Hmm, there is already an AWEB, written by U. Schweigert. I'm posting retrieval info RSN.

Yes, I'm familiar with Schweigert's "AWEB" but I have decided to use the name for several reasons. First, I don't think Schweigert's version has been supported in years. Second, I have been using the name for several years for an earlier version of web that I wrote several years ago, but never released. Third, it happens to be part of the name of my company, so it shows commitment to the product. In some circles that is important.

Finally, with my apologies to Schweigert, the version I have access to has problems. It may be that I have an old version or beta version of his AWEB, or that someone modified it before placing it on the archive I retrieved it from. Anyway, It doesn't support real Ada since it doesn't support Ada comments. Instead you must use @{, @} for comments. It doesn't support the allowable replacement symbols which are probably useful in Europe. His weave bunches statements together and could do a better job of formatting. I also ran into runtime errors on legitimate web programs.

However, Schweigert's weave did provide me with some ideas for my weave. His weave highlights Ada predefined identifiers, which gave me the idea to add that to my weave. I used his grammar as a starting point and then changed it significantly as I improved the formatting. (I doubt if I'm using more that 10 percent of his original grammar). There are a few other ideas I borrowed from his weave as well.

Actually, the introduction to my weave credits Schweigert, along with Wu and Baker, and of course DEK, Silvio Levy, Norman Ramsey, and John Krommes for influencing my Ada web system. Anyway, unless I hear from Schweigert that he is actively planning to support his version and bring it up to the level that I have brought mine, I will continue to call my version AWEB. For that reason I would appreciate not having Schweigert's version distributed.

If someone wants to use an Ada-based web immediately, try Norman Ramsey's outstanding Spider to generate an Ada web. It generates a weave and tangle for Ada, both of which are written in C. I know the University of Washington Unix TeX archive has Spider. I don't know if Norman still has his at princeton.edu.

If I remember correctly, there are some Ada limitations since Spider is a general purpose web generator. For example, I think there are some errors in its handling of lexical elements, which shouldn't be too difficult to fix. Another problem is the grammar. It handles much of Ada correctly but there are some places where the indentation isn't handled properly. Let me emphasize that it is not a problem with Spider, just the grammar, which you are free to modify.

If you want to use Spider's Ada web for now and my AWEB later, just use lowercase for all of the control codes and don't use the @' or @` control codes. Following those guidelines programs that weave and tangle correctly with Spider's Ada web should also do so with my AWEB, although the formatting will be different.

Mini-indexes in TeX: The Program

From: **Richard Walker**
Date: 31 Mar 1993

`TeX: The Program' and `METAFONT: The Program' have `mini-indexes' on the bottom right of each double page. How were they done? I weaved tex.web but I didn't get any of these indexes in the output. I'm surprised this isn't an FAQ. (Or am I just naive?)

From: **Stephan Eggermont**
Date: 31 Mar 1993

`TeX: The Program' and `METAFONT: The Program' have `mini-indexes' on the bottom right of each double page. How were they done? I weaved tex.web but I didn't get any of these indexes in the output. I'm surprised this isn't an FAQ. (Or am I just naive?)

Well, some time ago I asked the same question, so it might be... Anyway, the answer I got was: by hand. I suppose there is too much hand-tuning needed to get it working ok.

From: **Sanjeev Dharap**
Date: 31 Mar 1993

`TeX: The Program' and `METAFONT: The Program' have `mini-indexes' on the bottom right of each double page. How were they done? I weaved tex.web but I didn't get any of these indexes in the output. I'm surprised this isn't an FAQ. (Or am I just naive?)

I inquired about this a couple of years ago on comp.text.tex. They (especially Don Hosek) said that Knuth did them by hand.

From: **Silvio Levy**
Date: 31 Mar 1993

`TeX: The Program' and `METAFONT: The Program' have `mini-indexes' on the bottom right of each double page. How were they done? I weaved tex.web but I didn't get any of these indexes in the output.

Knuth has a program "ctwill" that works in two passes to make the mini-indexes. However, it takes a large amount of hand intervention, and for that reason he has not been distributing it. Some people on the list might have it, though.

From: **Bob Surtees**
Date: 31 Mar 1993

Knuth has a program "ctwill" that works in two passes to make the mini-indexes. However, it takes a large amount of hand intervention, and for that reason he has not been distributing it. Some people on the list might have it, though.

What does "large amount of hand intervention" mean. I would also be very interested in getting my hands on this program if it is available.

From: **Nelson Beebe**
Date: 31 Mar 1993

The story I got from David Fuchs, who wrote the code to do the mini-indexes on each page of Volumes B and D of Computers and Typesetting, was that the coding was a hack done just for those books, using a variety of non-portable tools to get the job done. I think those indexes are terrific, and keep hoping that someday, someone will write an extended set of changes for weave that will provide them.

From: **David Kastrup**
Date: 31 Mar 1993

Regardless of how they were done, they make the "added value" the books have over the free WEB sources. Note that unlike the TeXbook and the METAFONT book, which you are not allowed to reproduce the WEB sources can be made into printable form as much as you like (they are free!). But you will not get the little indexes that way.

From: **Richard Walker**
Date: 1 Apr 1993

\begin{naive} Would it be so hard to make the necessary modifications to weave and to cwebmac.tex? Actually, wouldn't LaTeX make the job easier? (Hint hint - I am eagerly awaiting CWEB 3.0.) \end{naive} Should we work towards producing these indexes? I read the first 125+ sections of TeX: The Program recently, and those mini-indexes made it just *so* much easier.

From: **Preston Briggs**
Date: 1 Apr 1993

Would it be so hard to make the necessary modifications to weave and to cwebmac.tex?

I don't see how it could be easy. Consider the problem for a second. Weave looks at the web and builds the .tex file, inserting necessary TeX directives, cross reference info, and the indices. But it doesn't know where TeX is going to make line breaks or page breaks. Therefore, it's going to have a hard time finding where to put mini-indices and what identifiers to put in the indices. Of course, we have a couple of examples that say it's possible, but we also have Beebe saying it required lots of hand tweaking. Much more

doable would be indices at the end of every major section. That way weave wouldn't have to know about page breaks.

From: **Norman Ramsey**
Date: 1 Apr 1993

I don't see how it could be easy. Consider the problem for a second. Weave looks at the web and builds the .tex file, inserting necessary TeX directives, cross reference info, and the indices. But it doesn't know where TeX is going to make line breaks or page breaks. Therefore, it's going to have a hard time finding where to put mini-indices and what identifiers to put in the indices.

That part's not hard; you use insertions, which are TeX's way of handling footnotes and floating figures. The hard part is the sorting and elimination of duplicates---these tasks are difficult to implement in TeX.

From: **Zdenek Wagner**
Date: 5 Apr 1993

I can see a way how to make the mini-indexes. Of course the mini-index must be built by TeX, not by weave. Wherever weave finds an item to be indexed, it should emit some TeX command. When TeXing the woven file, TeX should store the index items in the token register (or someone may find a better way). Then you have to rewrite the output routine which would typeset the mini-index from the token register and then clear it so that the token register starts again from scratch at the next page. Well, it seems easy but I know that the task is not trivial.

Literate Programming Using FrameMaker

From: **Stephen Cross**
Date: 1 Apr 1993

Here is a very quick and simple method for using FrameMaker to write literate programs: 1) Create two conditional_text tags (Special/Conditional Text...) Call the tags something like 'DocText' and 'CodeText'. 2) Apply the CodeText Tag to all the code in the document. 3) Apply the DocText Tag to everything else in the document. 4) To Tangle the document simply change the Show/Hide options for the conditional text so that only the CodeText is shown. 5) Save the Code to a text source file (File/Save As). Okay so its nothing special, but it does work. There are problably other (better) ways of doing this, any ideas?

From: **Lee Wittenberg**
Date: 2 Apr 1993

Stephen Cross writes: Here is a very quick and simple method for using FrameMaker to write literate programs: 1) Create two conditional_text tags (Special/Conditional Text...) Call the tags something like 'DocText' and 'CodeText'. 2) Apply the CodeText Tag to all the code in the document. 3) Apply the DocText Tag to everything else in the document. 4) To Tangle the document simply change the Show/Hide options for the conditional text so that only the CodeText is shown. 5) Save the Code to a text source file (File/Save As). Okay so its nothing special, but it does work. There are problably other (better) ways of doing this, any ideas?

It's a nice approach, but it isn't exactly "tangling," as the latter involves rearranging the code chunks from an human ordering (the sections in a web are supposed to be organized in a way that is easy for the human reader to grasp) into a machine ordering (something a compiler can accept). Your approach has the advantage of being able to interleave code and documentation, but it has the disadvantage that the code must still be presented in an order dictated by the machine. (cf. Dijkstra's "... I want the program written down as I can understand it, I want it written down as I would like to explain it to someone." ["Notes on Structured Programming", in _Structured_Programming_, Academic Press, 1972]).

From: **Sriram Srinivasan**
Date: 25 Oct 1993

I like the concept of literate programming (or what I understand of it). My understanding is: 1) Source and documentation is all present in one document. 2) The same document can be compiled, either for execution by a machine, or for publication. 3) The tools that implement literate programming provide a framework to achieve the above, and macros to format the stuff effectively. Unfortunately, all the examples that I have seen seem to use Tex as a base. I use FrameMaker for all my documents, and using Tex is simply too much effort devoted to formatting, IMHO. In the current set of tools, one requires a lot of "inband signalling" . That is, one has to have special symbols within the document that tag a certain paragraph as a piece of code, or as a part of the body, etc. The resulting document looks really messy, and it's painful to eyeball the document quickly.

I can achieve the same using FrameMaker or most other word processing tools, which allow you to tag paragraph types. So, I can have a paragraph template called "Code", that identifies the Code sections. Now, I can intermix graphics, tables, all kinds of formatting like automatic line numbering, cross-referencing, indexes etc. What I see on the screen is the final output, uncluttered by meta-information. Separating the code from the chaff is very easy, once a Frame file is converted to a MIF file (an awk script is given below). THE $(10^6) question: IS THIS IDEA WORTH EXPLORING FURTHER? I'd dearly like your comments on this subject. I am absolutely new to literate programming, and haven't read much beyond the FAQ, and some examples. I would think this can be achieved in say, Word for windows etc. which can save to an RTF format.

For those with access to framemaker, this is what I propose:

1. Have a paragraph tag called 'Code', and write all your code in this format.

2. Save as a mif file (or use fmbatch).

3. Run this thru the following awk script

```
BEGIN   {
                code = 0
        }

/<PgfTag \`Code\'/      {
                code = 1
                next
        }

/<PgfTag/ {
                code = 0
                next
        }

(code == 1) && /^ *<String \`/{
        # Ignore all other paragraph formats, and get only strings from
        # Code formats.

        # Need to strip this, for example ..
        # <String `printf (\xd2 Hello World\\n\xd3);'>

        # Strip junk in front
        sub(/^[ ]*<String `/,"")

        # Trailing junk
        sub(/'>$/,"")

        # Convert smart quotes to standard quotes
        sub(/\\xd2/,"\"")
        sub(/\\xd3/,"\"")

        print
        next
}
```

4. If you want pieces of code to go to different files, one can always have a special paragraph type ("File") that identifies a specific file type.

From: **Charles Bass**
Date: 25 Oct 1993

I think that your idea with framemaker is the only way to go. I looked at a package called ~ winwordweb that was a set of macros for Word for Windows that did what you are doing in Frame. In my opinion this is the nicest way to go because the front end is easy to use WSIWIG etc. The downside is that it is difficult for others to modify your literate programming program unless the have Frame. The tek based tools are (fairly) portable across unix and dos platforms. At any rate I like your idea of using Frame.

From: **Ralph Johnson**
Date: 25 Oct 1993

This is basically what the Literate Programmers Workbench from apple.com does. It is a fine idea.

From: **Lee Wittenberg**
Date: 26 Oct 1993

Sriram Srinivasan writes: Unfortunately, all the examples that I have seen seem to use Tex as a base. I use FrameMaker for all my documents, and using Tex is simply too much effort devoted to formatting, IMHO. In the current set of tools, one requires a lot of "inband signalling" . That is, one has to have special symbols within the document that tag a certain paragraph as a piece of code, or as a part of the body, etc. The resulting document looks really messy, and it's painful to eyeball the document quickly.

I can achieve the same using FrameMaker or most other word processing tools, which allow you to tag paragraph types. So, I can have a paragraph template called "Code", that identifies the Code sections. Now, I can intermix graphics, tables, all kinds of formatting like automatic line numbering, cross-referencing, indexes etc. What I see on the screen is the final output, uncluttered by meta-information. Separating the code from the chaff is very easy, once a Frame file is converted to a MIF file (an awk script is given below). THE $(10^6) question: IS THIS IDEA WORTH EXPLORING FURTHER? I'd dearly like your comments on this subject. I am absolutely new to literate programming, and haven't read much beyond the FAQ, and some examples. I would think this can be achieved in say, Word for windows etc. which can save to an RTF format.

Actually, what you propose is precisely the way WinWordWEB works, except that it uses internal Word macros instead of saving an RTF file. The idea of a CODE style paragraph that tools can recognize (and tangle and weave, as desired) seems to be the way to go. I recommend you forge on ahead. A lot of people have expressed interest in a Framemaker-based literate programming tool.

From: **Sven Utcke**
Date: 26 Oct 1993

Sriram Srinivasan writes: I like the concept of literate programming (or what I understand of it). My understanding is: 1) Source and documentation is all present in one document. 2) The same document can be compiled, either for execution by a machine, or for publication. 3) The tools that implement literate programming provide a framework to achieve the above, and macros to format the stuff effectively.

Well, my pet concept is missing: the possibility to write the code in any order I could dream off.

I can achieve the same using FrameMaker or most other word processing tools, which allow you to tag paragraph types. So, I can have a paragraph template called "Code", that identifies the Code sections. Now, I can intermix graphics, tables, all kinds of formatting like automatic line numbering, cross-referencing, indexes etc. What I see on the screen is the final output, uncluttered by meta-information. Separating the code from the chaff is very easy, once a Frame file is converted to a MIF file (an awk script is given below)

I certainly like the idea. Although I'm unlikely to ever use it myself (I'm hooked on LaTeX and wouldn't even consider using one of nowadays WYSIWYG-formater --- the output is much to ugly), it seems to be a good way to introduce more people to the concept of literate programming. If only the "chunk"-concept would be used (but that can't be so difficult). On the other hand: Isn't that what CLiP is doing?

From: **Bob Collins**
Date: 27 Oct 1993

Sriram Srinivasan recommends using FrameMaker for literate programming. However, Sriram Srinivasan did not mention all the advantages of FrameMaker. FrameMaker is, among other things, a hypertext document generator. One can construct (semi-automatically in FrameMaker 4) a set of hypertext links so that clicking on a hypertext word or picture will display related text. One can have the effect of a table of contents or index without page numbers. One can click on the "title" of a related code fragment to display the code fragment. Consider the primary advantage of a hypertext WYSIWYG document. One is always working with the latest version of the final document. No wasting of printer resources *or* working from slightly outdated printed documents.

FrameMaker also has an easy to use book concept with multiple chapters. This is the most logical way to break up a large document. FrameMaker will maintain and regenerate all sorts of links across documents. No need to write obscure macros. No need to modify these macros. FrameMaker allows all sorts of tagging (invisible to the eye but denoted in the bottom margin) of paragraphs and text. Uses of these can be so that lists can be generated and text extracted. This is how one can an automatic cross-reference and make it a hypertext cross-reference. There will no need to use cryptic marks in the text to indicate special things (like the continuation of the body of a code fragment. Use tags and a special typographic standard. Use variables to say "continued." All of these are easily feasible.

Sven Utcke objects that he would no longer be able to write code in an arbitrary fashion. Well, he may have misunderstood Sriram Srinivasan who claimed that *with almost no effort* one can use FrameMaker for literate programming. [Indeed, using conditional text for code, within FrameMaker one can view just the code using a simple dialog box. This requires no work at all.] Don't forget that the

Knuth system involves running programs to put code fragments together in the correct order. One can also write a program to do a similar thing to MIF output of FrameMaker documents. MIF is Maker Interchange Format, a TeX- or RTF- like marked ASCII text.

I have been reading this group for some time. I agree with all those who say the biggest drawback to literate programming is learning a new language -- TeX. TeX, written in the late 70's and popularized in the 80's is very much a 60's product in its interface. Granted it does things that were not done in the 60's. But it sure feels like JCL. Knuth, a wise and witty man, is a hacker at heart. He thinks obscurely and at a low level. He chooses low-level assembly language to describe algorithms. I cannot figure out how he has such a delightful sense of humor and an assembly language mentality. We seem to apotheosize the one person who least practices the things that make literate programming such a nice idea. [Knuth reminds me of a story (apocryphal?) about John von Neumann. At a party von Neumann was asked the puzzle: two trains 100 miles apart are approaching each other at 50 mph. A bumblebee, starting at one train, travels between trains at 25 mph, reversing directions when reaching an aproaching train. How far does the bumblebee travel before being crushed? von Neumann thought a short moment and promptly answered 25 miles. When asked if he knew the trick, he acted puzzled and said that he summed the infinite series.]

The *only* advantages of TeX that I know of are that it does mathematical typesetting nicely and it's free. FrameMaker does mathematical typesetting sufficiently well for literate programming.

From: **Sriram Srinivasan**
Date: 27 Oct 1993

Bob Collins writes: Sriram Srinivasan recommends using FrameMaker for literate programming. However, Sriram Srinivasan did not mention all the advantages of FrameMaker. FrameMaker is, among other things, a hypertext document generator. One can construct (semi-automatically in FrameMaker 4) a set of hypertext links so that clicking on a hypertext word or picture will display related text. One can have the effect of a table of contents or index without page numbers. One can click on the "title" of a related code fragment to display the code fragment.

[FrameMaker plug deleted ....] I have been reading this group for some time. I agree with all those who say the biggest drawback to literate programming is learning a new language -- TeX. TeX, written in the late 70's and popularized in the 80's is very much a 60's product in its interface.

You can say this again!

Now, for the downside of using Frame, or any other word processor. Note that I'm definitely NOT in favor of Tex still. 1) Speed of development - I am a power Emacs/Vi user, and Frame or WFW simply doesn't cut it in terms of development speed. 2) All my text processing tools can't be used - egrep, class browsers, awk, sed, sdiff, tags, and a zillion others. I managed to fix my mifToCode script to generate the correct line numbers, so that GDB isn't confused - but you see the point - I now have to worry about issues that weren't in text-land. 3) SCCS isn't very happy either. Have to move my stuff over to RCS, to be able to check in or out stuff. Either that, or have to convert all files to MIF before using SCCS. 4) Integrating this with other code in a production development environment seems to be a pain (I tried). (Other code - stuff already developed). 5) Using Tex and remaining in text-mode is a concept I find very painful. I do not want to think formatting when I am into development. Bob Collins has already said all I would have wanted to.

I guess I need a powerful argument for undergoing these hassles, especially the development cycle speed, before I really change my ways. Practically, how does it payback? Does it result in fewer edit/compile/run cycles? Has anyone used it for projects that have, say, 2 million lines of code, and has to be pushed out, say, yesterday? (Someone other than Knuth, that is!) Meanwhile, I convinced one of my friends to use my scripts and paragraph formats for a book he's working on. At the very least, we can test out the code that is going into the book by extracting all of it and compiling it. He's not completely in favor of doing development inside Frame yet.

From: **Joachim Schrod**
Date: 28 Oct 1993

Bob Collins writes: Knuth, a wise and witty man, is a hacker at heart. He thinks obscurely and at a low level. He chooses low-level assembly language to describe algorithms. I cannot figure out how he has such a delightful sense of humor and an assembly language mentality.

How? Simple: he's a scientist. How do you do *exact* analysis of run-time behaviour without an exact notion of the cost of basic operations? Give me the answer, and then we can see if we can get rid of MIX. Btw, Vol. 4 won't have MIX any more.

*The *only* advantages of TeX that I know of are that it does mathematical typesetting nicely and it's free. FrameMaker does mathematical typesetting sufficiently well for literate programming.*

How do you edit your FrameMaker documents on more than 40 different operating systems, ranging from 8086 PCs to Crays? How do you use FrameMaker in a convenient way as a background system? How do you use many languages? (Remember: Not everyone

writes in English...) Don't get me wrong: FrameMaker is surely one of the better commercial systems around -- but your evalution of the differences between TeX and FrameMaker is not even.

From: **Mark Friedman**
Date: 28 Oct 1993

From Joachim's response to Bob Collins' comments on Framemaker's strengths: How do you edit your FrameMaker documents on more than 40 different operating systems, ranging from 8086 PCs to Crays? How do you use FrameMaker in a convenient way as a background system? How do you use many languages? (Remember: Not everyone writes in English...)

I'll add 2 even "lower-end" systems to the editing list --- I often take notes & revise drafts on an _ancient_ Tandy T102 (not even MS-DOS, but it can spit out ascii), and I have 3 daughters (ages 11 to 18) that "share" an 80286 to (La)TeX documents created on a trio of (also ancient) Apple][']s. The point, of course is quite simple --- if it can produce ascii outout, _anything_ can be used for entry & editing.

As for the oft cited difficulty of learning LaTeX, my daughters have done _all_ their school reports since 3rd grade as well as lots of other "documents" (my 15 year old produces Girl Scout troop meeting minutes, plans & budgets), in English, French, German, Spanish and even ASL (American Sign Language --- a one-off report using a wonderful `hands' Metafont file). Overkill? Absolutely, but they happen to love it (they're even more critical of sloppy or ugly documents that I am, and I thought I was the perfectionist in the family). It has also led to various amusing incidents, like when my oldest was in 7th grade her English teacher stated that there was "only one right way to format a bibliograpy". Allyson's response was to take my copy of the Chicago Manual of Style to school, along with about a dozen versions of her assignment's bibliography, all formatted by different .bst styles, and ask which one was the "right" one...

From: **Bob Collins**
Date: 29 Oct 1993

This is my one response to my complaint about TeX and Knuth. I have the feeling this might turn into a jihad. FrameMaker is a commercial product suitable for producing large printed and hypertext documents, especially those that need maintenance and involve technical material and drawings. FrameMaker cannot do all things and some of what it does it does poorly. FrameMaker is not even my word processor of choice, but I use it because of its universality.

Joachim Schrod wrote: How do you edit your FrameMaker documents on more than 40 different operating systems, ranging from 8086 PCs to Crays?

FrameMaker is available on all the major platforms that can support it, that is, that have sufficient power and memory and that allow bitmap displays. This means most Unix systems, Macs, and MS Windows. Don't blame FrameMaker if it doesn't work on an AT. One cannot have its power for free. TeX is a filter -- all it does is translate from one form to another, like a compiler. Therefore it needs far fewer resources. FrameMaker integrates what TeX does, what your favorite editor does, what previewers do, what drawing programs do, and what dvi2lp utilities do. Face it -- how many of you wished that you could edit your preview when you found some error that you overlooked? How many of you preview on an 8086?

Joachim Schrod wrote: How do you use FrameMaker in a convenient way as a background system?

FrameMaker (on Unix) has both a macro system and a batch system. Both allow background processing. [In a silly moment I might ask, how do you allow foreground processing of TeX commands while you edit?]

Joachim Schrod wrote: How do you use many languages? (Remember: Not everyone writes in English...)

Last I checked, FrameMaker is available in 11 languages. (12, if you separate American English and Proper English.) This support is available in the dictionaries, thesaurus, system variables (like dates), and numbering schemes.

Mark Friedman wrote: I'll add 2 even "lower-end" systems to the editing list --- I often take notes & revise drafts on an _ancient_ Tandy T102 (not even MS-DOS, but it can spit out ascii), and I have 3 daughters (ages 11 to 18) that "share" an 80286 to (La)TeX documents created on a trio of (also ancient) Apple][']s. The point, of course is quite simple --- if it can produce ascii outout, _anything_ can be used for entry & editing.

One can import text into FrameMaker. FrameMaker will even get rid of extraneous spaces and turn straight quote marks into curly ones while it imports (if you want). In addition, FrameMaker supports MML import. MML stands for Maker Mark-up Language and is a

SMGL-like form for typographic conventions. So your kids can edit their instructions in their text, read it into FrameMaker, and then edit the preview-like display.

Mark Friedman wrote: As for the oft cited difficulty of learning LaTeX, my daughters have done _all_ their school reports since 3rd grade as well as lots of other "documents" (my 15 year old produces Girl Scout troop meeting minutes, plans & budgets), in English, French, German, Spanish and even ASL (American Sign Language --- a one-off report using a wonderful `hands' Metafont file). Overkill? Absolutely, but they happen to love it (they're even more critical of sloppy or ugly documents that I am, and I thought I was the perfectionist in the family). It has also led to various amusing incidents, like when my oldest was in 7th grade her English teacher stated that there was "only one right way to format a bibliograpy". Allyson's response was to take my copy of the Chicago Manual of Style to school, along with about a dozen versions of her assignment's bibliography, all formatted by different .bst styles, and ask which one was the "right" one...

Well, Mr. Friedman has a smart set of kids. I congratulate them and him. I have my own set of computer-literate kid stories, and I will be pleased to share them. I wish all children had entre to computer-wise parents and equipment at the earliest age. It would help me as a university professor. [My favorite kid story is how my son, when 5, learned how to use a paint program by memorizing what the menus did. He could not read, but he knew the alphabet.]

However, I have gone through the hassle of trying to help graduate students try to write using LaTex. The learning curve is steep. In fact, some of our faculty couldn't figure out how to get rid of a space before a grammar production in our comprehensive examination question list. The number of macros (written by different folks) confused the issue. The smartest people can get turned off by TeX. Is literate programming closed to them?

Let's get serious, however. For those who know, TeX works great. Just like punch cards for editing. However, most people found interactive editing much easier. And those who were used to punch cards sometimes found it difficult or unnecessary to change. And they were right. You can do anything you want with a punch card. Make a wish list for literate programming. Don't worry about how it can be accomplished, or how the typesetting is done. I think that most of us would like to edit the finished product rather than the marked-up product. [I would like to do it by voice, e.g., Bob Collins: Show me the code fragment for the input loop. Computer: Did you mean code fragment or chunk?] At the least, this means interactive TeX. Let's aim for that. Meanwhile, I think that FrameMaker offers a reasonable alternative. These are my final words. Flame on.

From: **Jim Rudolf**
Date: 29 Oct 1993

Charles Bass writes: I think that your idea with framemaker is the only way to go.

Personally, I think using FrameMaker is a good idea, but not necessarily by *editing* in FrameMaker. We are just a bunch of programmers who use vi and work fast in it. We want to be able to document our code too, and embedding the doc in the source code (keeping in mind the readability of the source for us programmers, of course) means we are more likely to update the doc when we update the code. Our solution was to write a couple perl scripts to do documentation generation for us. One script scans the source file for a few keywords that can describe the entire module or individual procedures. Once all the info for a file is collected, another set of scripts will create a file either in man page or MML format. The two formats are quite different, as the man page format is used internally, and the MML format is used to make a FrameMaker book for customers. Other back-end scripts could easily be written when needed. We've found this to be a good combination for us. Has anyone else had any experience (good or otherwise) with this approach?

From: **Michael Koopman**
Date: 29 Oct 1993

Bob Collins wrote: This is my one response to my complaint about TeX and Knuth. I have the feeling this might turn into a jihad. FrameMaker is a commercial product suitable for producing large printed and hypertext documents, especially those that need maintenance and involve technical material and drawings. FrameMaker cannot do all things and some of what it does it does poorly. FrameMaker is not even my word processor of choice, but I use it because of its universality.

I don't believe you'll see an uprising from your comments on TeX. I have not heard anyone pining over the punch card machine nor against an interactive TeX editor/previewer. The TeX hacks promote the capability, portability and succintness of the language. Whether or not FrameMaker is as clever as TeX seems somewhat irrelevant. If you want a FrameMaker web then what qualities of the web are critical and which of these 'requirements' are met by FrameMaker and not met by TeX. That is, why does literate programming in FrameMaker detract from literate programming in TeX? Does FrameMaker read TeX input, or more importantly, can FrameMaker read a CWEB, FWEB, etc. web and not munge it beyond recognition? Let's not hack each other up in religious wars over the "right" editor or typesetting mechanism. Let's focus on the benefits and needed improvements in the literate programming techniques at hand, instead. Peace be with the Literati.

From: **Osman Buyukisik**
Date: 29 Oct 1993

People seems to be missing the main point about literate programming. It is not just a documentation scheme! You start from the design phase and use whatever form of editor/TP you want to explain the code and write the code in a way that is independent of the whims of the compiler that you will use. In the end, the final product should resemble a `book'. Because of these points I don't think c2man and others like it are literate programming tools. You can use framemaker/interleaf but someone has to write some expansion to it so that `tangle' command is available in the menu (like the WinWordWeb). No one is forcing TeX/LateX but these tools are FREE and widely available. People should not be flaming each other instead if you want a framemaker literate programming tool just create it and announce it to the net. For myself I cannot see how hypertext helps since I like the printed output! I use interleaf at work and all the formulas and graphs do not look the same on the screen as printed on paper, and are hard on the eyes.

From: **Uttam Narsu**
Date: 7 Apr 1994

Does anyone know of any Web (literate programming) tools for FrameMaker? We need to be able to keep mathematical equations, graphics and code together. I've taken a look through the FAQ, but didn't see anything that would really help us. Most Web tools won't be acceptable to our programmers because the tools demand some knowledge of TEX (especially for formatting equations and pictures). Since Frame is already used for much of our documentation, it seems to be a natural choice for a front-end to a Web.

The FAQ mentions two tools that would almost be useful: CLiP and WordWeb. Alas, CLiP is written in ISO Pascal, which is useless to me since I don't have a Pascal compiler for my SGI. WordWeb is completely dependent on Microsoft Word, so no help there. As a last resort, if anyone knows of any filters between TEX and Frame MIF files, that would be helpful.

From: **Jacob Nielsen**
Date: 7 Apr 1994

Does anyone know of any Web (literate programming) tools for FrameMaker?

Some time back, there was someone who had done literate programming using FrameMaker, anyone got that posting?. In the meantime take a look at nuweb. You can tell nuweb to only produce the code files and *not* output the documentation. The downside is that you have to look at scraps like this in the FrameMaker document:

```
@o test.c
@{
  @<Include files for the test@>
  @<Defines for the test@>
  @<All the rest...@>
@}
```

If you can save the FrameMaker document, so that you have text like this, you should be home free using nuweb :-)

From: **Sriram Srinivasan**
Date: 9 Apr 1994

For those with access to framemaker, this is what I propose:

1. Have a paragraph tag called 'Code', and write all your code in this format.
2. Save as a mif file (or use fmbatch).
3. Run this thru the following awk script.

```
BEGIN   {
        code = 0
    }

/<PgfTag \`Code\'/  {
        code = 1
        next
    }

/<PgfTag/ {
        code = 0
        next
```

```
    }

(code == 1) && /^ *<String \`/{
        # Ignore all other paragraph formats, and get only strings from
        # Code formats.

        # Need to strip this, for example ..
        # <String `printf (\xd2 Hello World\\n\xd3);'>

        # Strip junk in front
        sub(/^[ ]*<String `/,"")

        # Trailing junk
        sub(/'>$/,"")

        # Convert smart quotes to standard quotes
        sub(/\\xd2/,"\"")
        sub(/\\xd3/,"\"")

        print
        next
    }
```

4. If you want pieces of code to go to different files, one can always have a special paragraph type ("File") that identifies a specific file type.

From: **William Clodius**
Date: 18 Oct 1994

I am interested in the potential for literate programming, but most literate programming tools appear to rely upon TeX or LaTeX. The team I am involved with are doing most of their documentation with FrameMaker. Does CLiP or some other tool facilitate literate programming with FrameMaker?

From: **Eric van Ammers**
Date: 25 Oct 1994

Regarding your mail of on the question if CLiP can be used with Frame Maker, I expect no problem. CLiP can be used in combination with any system which provides for a decent ASCII-export.

Future developments

From: **Anthony Coates**
Date: 08 Apr 1993

Paul Lyon writes: On a related point, it seems to me that, at least at present, the resources, in the way of time and effort, available to improve the state of literate programming tools, are rather limited. We are still waiting for the new version of CWEB , for example; I presume that this is because Levy and Knuth have little time to devote to it. Accordingly, it seems to me that such energy and time as we have to spare ought to go into improving the tools we already have rather than into the development of GUI based ("WYSIWYG") literate programming tools. I, for one, was disheartened to read Lee Wittenberg's posting about his beginning efforts using Word for Windows, and that by S.C. Cross on using FrameMaker. Besides a better interface to pertinent TeX or LaTeX macro packages, I can think of several things that would be of aid to the literate programmer, such as better macro processing---to be achieved by stealing as much as seems applicable from GNU M4 (and perhaps also the macro capability in the preprocessor for the COOL C++ library, and other places as well), or cross-indexing over multi-module web source, and the like.

Disheartened? This intrigues me. After all, few of us have the foresight to really know what is going to be the way of the future. I for one don't see that improving the tools used by an elite few (and we are, let's face it, a minority) should take precedence over creating tools that could be more acceptable to the programming community at large. Fewer and fewer people use command lines any more, and Emacs is still often only for the cogniscenti, wonderful as it is. Many of my colleagues write all their scientific papers in Microsoft Word; if I was to try to convince them to use Literate Programming tools, I would have no chance with an Emacs/TeX combination. I might have a small chance with a Word for Windows package; who knows, using the coming abilities of the OLE 2.0 (Object Linking Environment) one could create a hypertext-style environment that might be much more attractive than anything currently seen under

UNIX/TeX/Emacs (at least for now, we'll see what Sun can offer in terms of objects and such in the future). OK, so the argument is that maybe there are too few of us to chase all of these goals at once. But then, maybe there are too few of us to yet come to a consensus on what is best. We will see.

From: **Paul Lyon**
Date: 18 Apr 1993

Tony Coates writes: Disheartened? This intrigues me. After all, few of us have the foresight to really know what is going to be the way of the future.

To be sure, but then I was not thinking about the next few years so much as I was thinking about the next few months, or perhaps a year or so. It seems unlikely there will be but one "way of the future", and in any case I do not expect that literate programming will ever be more than a minority pursuit (see below).

I for one don't see that improving the tools used by an elite few (and we are, let's face it, a minority) should take precedence over creating tools that could be more acceptable to the programming community at large. Fewer and fewer people use command lines any more, and Emacs is still often only for the cogniscenti, wonderful as it is. Many of my colleagues write all their scientific papers in Microsoft Word; if I was to try to convince them to use Literate Programming tools, I would have no chance with an Emacs/TeX combination.

The urgency of spreading literate programming to the programming community at large escapes me. For one thing, one would have to foster better writing skills while one was at it. This pessimism on my part may be parochial: some of the courses I teach here are courses "with a substantial writing component", as it is put hereabouts. The University wisely requires that all its undergraduates take such courses in addition to the basic English composition course; judging by the overall quality of the work I receive, the undergraduates are much in need of practice and assistance. At that, the students I see are mostly Liberal Arts students, and 3rd and 4th year ones to boot.

No doubt there is a significant non-overlap of the set of programmers possessed of decent writing skills with the set of those prepared to cope with TeX or LaTeX. Your colleagues may well be instances to the point. But then, one of the improvements I would hope for in the existing tools would be a better interface to TeX or LaTeX. Surely we can reach some agreement among us about what is desirable along these lines? The other thing needed to bridge the gap, so it seems to me, is a decent online help system for TeX or LaTeX, the which could be extended to include Web commands. Embedded command formatting would not be so difficult to deal with were one not constantly forgetting which command sequence does what. A main advantage of the GUI-based word processors is that they reduce the need for having the formatting manual (the TeXbook, in my case) by one's side.

As for the gap between the text editors embedded the GUI-based word processors with Emacs, surely the various X windows versions of Emacs can serve as a bridge? Having posed this, I should note that I have never actually used Epoch or Lucid Emacs. The Emacs port to OS/2 2.0 is what I have to work with, and a Presentation Manager version of that will not soon be available, alas. [Emacs Version 19 will be a precondition for such, but one grows weary waiting for that...] Again, though, it seems to me that the major advantage of the GUI-based editors is the menu/dialog box system, which reduces the load on one's memory. Emacs could surely use some of that! (The Emacs 18 help system is clumsy at best; again, I mostly find myself reaching for the printed manual.)

But I suppose this is mostly by-the-by; even with such improvements as as these, I doubt you would be able to wean your colleagues away from what they are using to consider tools that require Emacs plus TeX, or some such combination. But then I wonder, what is so terribly wrong with this state-of-affairs? Is it that you must use or maintain code that they write? Or that they expect to use code that you write? Trying to use literate programming tools on a multi-programmer project when none of the others are, will, I agree, pose a significant problem.

I might have a small chance with a Word for Windows package; who knows, using the coming abilities of the OLE 2.0 (Object Linking Environment) one could create a hypertext-style environment that might be much more attractive than anything currently seen under UNIX/TeX/Emacs (at least for now, we'll see what Sun can offer in terms of objects and such in the future).

I am unfamiliar with OLE 2.0. But I wonder what the impact of hypertext features would be on tangling and weaving. The hypertext model is, after all, a general graph. The compiler and the formatter, however, expect linear text in a fairly rigid order. What one could do, I suspect, is to include the hypertext links in the web source to make navigation among the parts of it easier when writing, but leave the undelying text in the proper linear order, then have the tangle and weave processors strip the links out before handing the results on to the compiler and formatter.

OK, so the argument is that maybe there are too few of us to chase all of these goals at once. But then, maybe there are too few of us to yet come to a consensus on what is best. We will see.

I think that there are enough of us to see what is better, if not what is best :-) We all realize, for example, that user configurable pretty-printing would be desirable in CWEB , FWEB, and the like. And anyone who has stared at the code for CWEB , FWEB, and SpiderWeb, should realize that the existing code is not readily extended. Perhaps the first thing that should be done with CWEB 3.0, when it is released, is to rewrite it in C++, with appropriate care taken to do a proper "object-oriented" job of it :-)

From: **Stephen Fulling**
Date: 18 Apr 1993

Paul Lyon writes: The urgency of spreading literate programming to the programming community at large escapes me. For one thing, one would have to foster better writing skills while one was at it. This pessimism on my part may be parochial: some of the courses I teach here are courses "with a substantial writing component", as it is put hereabouts. The University wisely requires that all its undergraduates take such courses in addition to the basic English composition course; judging by the overall quality of the work I receive, the undergraduates are much in need of practice and assistance. At that, the students I see are mostly Liberal Arts students, and 3rd and 4th year ones to boot. No doubt there is a significant non-overlap of the set of programmers possessed of decent writing skills with the set of those prepared to cope with TeX or LaTeX.

All the more reasons why all students should be introduced to literate programming and to TeX as early as possible! At present they see a total disconnect between "verbal" courses and "mathematical" courses (and activities and occupations). They also see precious little connection between computing and other technical courses (e.g., calculus). It should all be a seamless web. Maybe it's parochial optimism, but I have the impression that when you force them to TRY, most students produce much better written work than their usual output.

From: **Lee Wittenberg**
Date: 19 Apr 1993

Paul Lyon writes: The urgency of spreading literate programming to the programming community at large escapes me. For one thing, one would have to foster better writing skills while one was at it. This pessimism on my part may be parochial: some of the courses I teach here are courses "with a substantial writing component", as it is put hereabouts. The University wisely requires that all its undergraduates take such courses in addition to the basic English composition course; judging by the overall quality of the work I receive, the undergraduates are much in need of practice and assistance. At that, the students I see are mostly Liberal Arts students, and 3rd and 4th year ones to boot.

I find the same situation with my students, although I teach mostly CS students. We also have "writing emphasis" courses (I'm trying to develop one in Literate Programming, e'en as we speak). One of the things I've found with literate programming, though, is that a poorly written literate program seems to be easier to maintain than an average quality non-literate one. Admittedly, all my evidence is anecdotal, and I have yet to see a really horrible literate program (perhaps once I've got a bunch of students trying it?), but maintenance is such an important part (perhaps the most important part) of CS, that literate programming seems worth pursuing for that reason alone.

No doubt there is a significant non-overlap of the set of programmers possessed of decent writing skills with the set of those prepared to cope with TeX or LaTeX. Your colleagues may well be instances to the point. But then, one of the improvements I would hope for in the existing tools would be a better interface to TeX or LaTeX. Surely we can reach some agreement among us about what is desirable along these lines? The other thing needed to bridge the gap, so it seems to me, is a decent online help system for TeX or LaTeX, the which could be extended to include Web commands. Embedded command formatting would not be so difficult to deal with were one not constantly forgetting which command sequence does what. A main advantage of the GUI-based word processors is that they reduce the need for having the formatting manual (the TeXbook, in my case) by one's side.

Agreed, although I think that "agreement...about what is desirable..." is one of those things that never happens, for various reasons. Corporations want standards, but they want everyone else to accept their standard rather than accept someone else's (NeXT being a notable exception) -- witness the current PC-database-standard wars between Microsoft and Borland (and the various companies behind each one), not to mention the upcoming operating system war. Regarding "embedded command formatting," I've noticed that everyone who uses WinWord around here runs it with all of the formatting symbols (paragraph marks, dots for spaces, etc.) displayed. If they don't, WinWord tends to do strange things when they delete invisible "characters." Is this WYSIWYG or embedded commands?

One problem that surfaces with WYSIWYG editors, but not text-editor/embedded-command systems is one of tools. Most WYSIWYG editors have undocumented (usually binary) file formats, so we can't rely on the many text-based tools that have been developed (e.g. most everything in UNIX). We have to rely on the manufacturers to supply new tools, or work out the file format (ugh!) and make our own. One of the things I like about the various WEB systems (and TeX as well) is that they are maintained by people who actually use them on a regular basis, which is _not_ true for (I would venture to say) most commercial software.

As for the gap between the text editors embedded the GUI-based word processors with Emacs, surely the various X windows versions of Emacs can serve as a bridge? Having posed this, I should note that I have never actually used Epoch or Lucid Emacs. The Emacs

port to OS/2 2.0 is what I have to work with, and a Presentation Manager version of that will not soon be available, alas. [Emacs Version 19 will be a precondition for such, but one grows weary waiting for that...] Again, though, it seems to me that the major advantage of the GUI-based editors is the menu/dialog box system, which reduces the load on one's memory. Emacs could surely use some of that! (The Emacs 18 help system is clumsy at best; again, I mostly find myself reaching for the printed manual.)

I work with MS-Windows rather than OS/2, but I echo Paul's wish for a Presentation Manager (Windows in my case) version of Emacs. I've tried the Micro Emacs Windows port, and while it's really good, there are a few things (e.g., not highlighting mouse-selected text) that make it difficult to use with any efficiency.

But I suppose this is mostly by-the-by; even with such improvements as as these, I doubt you would be able to wean your colleagues away from what they are using to consider tools that require Emacs plus TeX, or some such combination. But then I wonder, what is so terribly wrong with this state-of-affairs? Is it that you must use or maintain code that they write? Or that they expect to use code that you write? Trying to use literate programming tools on a multi-programmer project when none of the others are, will, I agree, pose a significant problem.

Emacs plus TeX, of course, gives you more control than does a WYSIWYG word processor, but then again, most people don't seem to want that kind of control. I reckon Paul's right, here, too.

I think that there are enough of us to see what is better, if not what is best :-) We all realize, for example, that user configurable pretty-printing would be desirable in CWEB , FWEB, and the like. And anyone who has stared at the code for CWEB , FWEB, and SpiderWeb, should realize that the existing code is not readily extended...

I kind of take exception to this statement (in a nice way, of course :-)). While I haven't looked at the code for FWEB, I _have_ worked with both CWEB and Spidery WEB, and was quite astonished at how easy they were to understand and modify. I had to fix several bugs before I could CWEB working on my PC (all fixed now in the forthcoming 3.0 release), and have made several major extensions to my copy of Spidery WEB. All of these were surprisingly easy, and did not break the existing code.

From: **Mark Purtill**
Date: 20 Apr 1993

Agreed, although I think that "agreement...about what is desirable..." is one of those things that never happens, for various reasons. Corporations want standards, but they want everyone else to accept their standard rather than accept someone else's (NeXT being a notable exception)

Sorry, have I missed a NeXT announcement of some sort? Last time I looked, NeXT was hardly an exception: Display PostScript rather than standard X11, Objective C rather than standard C++. True, they're porting this non-standard software to a standard CPU (i486), but not to one of the standard OSs, DOS, Windows, OS/2, Linux :-), but as a new, non-standard OS. Or do you mean that NeXT is an exception to "corporations want standards"? That I'd believe.

From: **Lee Wittenberg**
Date: 20 Apr 1993

Mark Purtill writes: Sorry, have I missed a NeXT announcement of some sort? Last time I looked, NeXT was hardly an exception: Display PostScript rather than standard X11, Objective C rather than standard C++. True, they're porting this non-standard software to a standard CPU (i486), but not to one of the standard OSs, DOS, Windows, OS/2, Linux :-), but as a new, non-standard OS. Or do you mean that NeXT is an exception to "corporations want standards"? That I'd believe.

What I meant was that, rather than invent their own "standards," the NeXT people looked around and said, "What's already out there that we can adopt as a standard?" They decided (rightly or wrongly, possibly both) on Unix as the operating system, PostScript as the display language, and Objective C as a systems language. There's no way of knowing for sure but I believe their reasons were something like:

1. Unix because, let's face it, there isn't any other operating system that is implemented on a large number of different platforms.

2. Display Postscript so that they could use the same graphics language for printer and screen output. X-Windows is lovely, but it would probably be a lot tougher to build an X11 printer than it was to build a PostScript video display.

3. At the time the NeXT came out, Objective C and C++ were pretty much running neck and neck as regards which

> language was C's "legitimate object-oriented heir." C++'s current popularity seems to be as much from media attention as anything else, but the choice was not clear cut at the time. NeXT flipped a coin, and backed the wrong horse, although the decision to go with an object-oriented descendant of C accurately predicted the current trend. [Actually, Objective-C still has its adherents. Baby duck syndrome aside, whether you prefer Objective-C or C++ depends mostly on whether you come from the Smalltalk or the Simula school of object-oriented programming, the former advocating type checking at run-time, the latter, at compile time. Most of us have strong opinions on this issue, but this forum really isn't the place to get involved in that religious war -- we have enough of our own.]

The main point I was railing at was the NIH (Not Invented Here) problem that is rampant in our industry. I chose NeXT as an example because all of their choices involved "standards" developed elsewhere that were already (somewhat) widely in use. I apologize for not making myself clearer the first time.

From: **Bart Childs**
Date: 20 Apr 1993

Lee W's comments are well founded. He mentions anecdotal evidence. Has anyone done any careful measurements of this type? I keep hoping to find a graduate student who will want to but one really needs funding to be able to support controlled studies with parallel control populations...

Lee is a bit more optimistic about the file formats of WYSIWYG editors. We have worked on conversion programs from various editor formats to TeX (or LaTeX). It is not correct to call them undocumented; the documentation is often out of date (as most seems to be) or more often just nearly impossible to find. His sense was correct, but the biggest problem we found was that the owner RESERVES THE RIGHT TO CHANGE IT AT ANY TIME WITH NO NOTICE. Indeed, the worst case implied by such a statement has been done and only after you install the new version do you learn that the binary format has changed. Have you ever used WORD to wander around between DOS, MACs, and NeXTs? Even using RTF you don't have a panacea of portability. Knuth said one of his design goals for TeX was that a real "archive" would be possible. TeX written more than 10 years ago still runs. (It has been extended but not really changed.)

I long ago quit trying the evangelist business. I don't try to convince people that they should use TeX instead of a WP. Most of them do a more than adequate job on memoranda, ... Human nature still dominates and it is rare that I accept statements of the nature that "(any) WP does a good job of formatting a document with mathematics (or a large number of othere difficult items)." Granted a lot of people say nobody complained ... The majority of the customers at MacDonalds don't complain but they also don't try to convince others that it is gourmet food ... My compliments to Lee, Paul, and others for a really good set of comments.

From: **Paul Prescod**
Date: 20 Apr 1993

The main point I was railing at was the NIH (Not Invented Here) problem that is rampant in our industry. I chose NeXT as an example because all of their choices involved "standards" developed elsewhere that were already (somewhat) widely in use. I apologize for not making myself clearer the first time.

Care to explain NeXTMail? And what about X? I get the impression that NeXT Corp. would not add X to NeXTStep even if you programmed it up for them. What they have is better, so what already exists is irrelevant! Even if they could work well together. I wonder how standard their Unix implementation is too.

From: **Lee Wittenberg**
Date: 22 Apr 1993

Paul Precod writes: Care to explain NeXTMail? And what about X? I get the impression that NeXT Corp. would not add X to NeXTStep even if you programmed it up for them. What they have is better, so what already exists is irrelevant! Even if they could work well together. I wonder how standard their Unix implementation is too.

1. Dunno about NeXTMail. It seems to handle plain ol' Internet mail just fine (unfortunately our NeXT is in NJ and I'm in TX, so I can't check out any of this -- please take this and the following comments as my impressions rather than gospel truth).

2. I was under the impression that NeXTstep _did_ support X (I think I read it in the documentation somewhere). Maybe not.

3. Their implementation of Unix is as standard as any I've seen. In fact, when I ftp something from the net, I usually try to build it on the NeXT first, because it almost always builds correctly the first time. I occasionally have problems on other Unix machines. [Relevant side issue: Now that C is officially standardized, I find it interesting that none of the cc compilers on any of the Unix systems I have accounts on conforms to the standard.]

4. I didn't claim the NeXT people were perfect -- just that they did a few major things right IMHO :-).

From: **Paul Lyon**
Date: 22 Apr 1993

Lee Wittenberg writes: I find the same situation with my students, although I teach mostly CS students. We also have "writing emphasis" courses (I'm trying to develop one in Literate Programming, e'en as we speak).

Well, I like this idea! A couple of suggestions, if I may. Firstly, you might want to look at Chapter 9 ("Mathematical Writing") of Knuth's "Literate Programming", if you have not already done so, for some hints about how students at Stanford got on with literate programming. I presume you are considering one or both of Knuth's book and Wayne Sewell's "Weaving a Program" as texts. I am tempted to suggest that if you feel that you can only use one, then Knuth is the one to try, even though not all of it is about literate programming. It seems to me that you are more apt to get them thinking about style and organization of their writing this way.

It seems to me that one ought to play it straight here, namely to try to get the students to follow the conventions of expository writing as best they can, allowing for the differences between a literate programme and other types of exposition writing. They should give a proper introduction explaining to the reader what is coming, try to follow a coherent narrative organization, ond so on. You are, after all trying to help them improve their writing, albeit in a unusual genre of exposition.

One interesting question about style is this: should they try to write a proper conclusion to their programme summing up what it does and what they think is most important among the ideas for the programme they have previously discussed? The examples of literate programmes we have from Knuth do not have a conclusion. More generally, it seems to me that in literate programmes written in noweb or Funnelweb, as they allow a more free form style, one may be able to use a model of expository writing closer to the traditional models, than with those written in CWEB , FWEB, or a Spidery Web. A proper conclusion may or may not be useful, but one perhaps should try it to see.

In general, it seems to me, that since this is meant to be a writing course, the students should include more by way of commentary in what they write than they might in a literate programme written for use, and you should insist on good organization and writing style. The problem of expository structure in a literate programme raises a second issue. Though they are certainly an improvement on manuscript and typewriting, conventional text editors, including Emacs, do not offer that much by way of support for crafting expository organization nor do they provide much help for refining the ideas to be presented. For that sort of thing, one needs an "outliner". Unfortunately, the only ones that I know about are for MS-DOS or the Macintosh. Indeed, the only two I have tried are both DOS programmes, viz. MaxThink and Kamas.

I have used MaxThink for some years now in writing my lecture notes, and am quite pleased with it. Now I find myself trying to use it for literate programming as well. The idea is to think through the ideas for the programme or programme or library modules, writing these in as topics in the outline, adding source as appropriate as text attached to a topic, then use the structuring features of MaxThink to work out an organization for the web source, and finally have MaxThink write it out in a suitable form to dump it into Emacs to finish the job. Pity that time constraints have prevented me thus far from completing this process for any of the things I am working on :-)

As for Kamas (which is a DOS shareware product), I have only recently downloaded it from one of the mirrors of the simtel20 archive (e.g. ftp.uu.net, ~ftp/systems/ibmpc/msdos/simtel20/editor/kamas25.zip), and have only played with it a bit, so I can't say much about it, sawe that it seems to offer an interesting alternate approach to MaxThink, and also seems to have some of the more useful features of MaxThink, though not, by any means all. I should note that I also downloaded PC-Outline while I was at it (this is, curiously, in the txtutl subdirectory of the simtel20 archive---pco334.zip), but it seems to me that PC-Outline lacks most of the amenities of MaxThink and Kamas, and is decidedly less useful as a consequence. Yet even PC-Outline is clearly more useful than the best Emacs package, namely "allout.el", and never mind the equally inferior outlining components of Word and Wordperfect. (I just downloaded web-mode.el, and notice that it too offers some outlining capabilities, but no more, so it seems to me than "allout.el". To be sure "web-mode.el" looks to be well thought out and written; Bart Childs and Mike Motl have done a nice job here.)

Having said this, I should note that I took the trouble to investigate alternatives to MaxThink so as to be able to present alternatives to the students in the "writing component course" I am teaching this term, thinking that use of such a tool could help them with the problems of expository organization that I saw in many of their papers. My presentation of this idea to the class some weeks ago, however, seems to have gone over like a lead ballon. I was pleased to discover, though, through a conference with a student in that class several days later, that her paper, which was unusually well organized, had been written with the aid of the outline capability in MS-Word. Primitive though that is, she had made good use of it in structuring her exposition.

I am also moved to note a remark that Neil Larson, the designer of MaxThink, included in the user manual, namely: "Word processing focuses on WYSIWYG---what you see is what you get", but "MaxThink focuses on WYTIWYG---what you think is what you get". Later on in the manual, after remarking on the work it would take to convert MaxThink from a text-mode DOS programme to MS-Windows, he explains his decision not to undertake this thusly: "After all this, the intellectual processes for better thinking, writing, and planning are not improved in the slightest. Summary: While Windows may be fashionable, it does not extend MaxThink's capabilities!" (Apologies, sort of, to the WYSIWYG fans, but this is a point worth pondering in that connection, especially since Leslie Lamport makes a similar point, though with rather less justification, early on in his LaTeX book.)

In any event, I am not actually recommending that one require students to lay hands on a good outliner, but merely trying to make a few points about appropriate tools for expository writing in general, and literate programming in particular. But I do commend Lee Wittenberg's idea to others reading this list who teach CS students, and I would encourge Lee to post information about the design for

his course to the mailing list. (I have more things to say about other matters of interest that Lee Wittenberg raises in his thoughful comments, but I will do this in a subsequent post.)

Experiences with literate programming so far

From: **Kayvan Sylvan**
Date: 13 Apr 1993

Whew... I just spent about a day and a half reading the entire litprog archives. I figured it would be good if I were a bit more informed before I started sending to this list. I'm new to Literate Programming, though I've known about it for some years now. Recently, I had the pleasure of writing a real-world literate program for a client (and then an in-house tool to aid in a large and nontrivial global change to a massive amount of 4GL source code).

The first program that I wrote, dbstrip, took about two days to write, and about an hour or two to find and fix the one (fingers crossed \smiley) bug. I originally wrote it in CWEB 2.8 but switched over to FWEB (for reasons I'll talk about later). The purpose of the dbstrip program is to strip unwanted characters from STRING fields in a Unify database. I wrote this program using WEB mostly as an experiement to save time and effort (now and later). Part of my task was to write a design document, have it approved by my client, and then write the actual program. Using Literate Programming, I was able to produce an early version of the program/document which showed the essentials of my design, have it approved by the client, and then go on to complete the project (and have a beautifully typeset technical document as a side effect which I then delivered along with the code). It also made the task of coding much more fun. :-)

To continue my tale, I now had a working program of high quality, produced in less time than it would take to create and debug a conventional program. I was ecstatic... but... ;-) Delivering the tangled code to my client proved a bit problematic. CWEB's ctangle produced uncommented, unformatted, unreadable C code. My client was not happy with this. I ran the code through indent and that was a bit more palatable, but the stripping of my copious comments was still annoying. At this time, I ran into FWEB, and since I was also looking for ANSI C/C++ support, I decided to try using FWEB. I brought it up on my ESIX system (with only a minor bug fix) using gcc-2.3.3. I converted my dbstrip program from CWEB to FWEB, and found, much to my pleasant surprise, that FWEB was much more customizable than CWEB. As a side comment, John Krommes, the author of FWEB, has been very helpful and responsive to Email and worked with me on finding solutions to various problems. Anyway, I regenerated the tangled code using ftangle, with the "-v" command line option that tells ftangle to keep the comments, ran the code through indent, and delivered the final code (along with the web file and a pointer to ftangle) to my client who was happy with the code and impressed by the documentation.

The saga doesn't end here. :-) Upon porting this simple program to the RS6000, I ran into a problem and I'm wondering if anyone has seen anything similar to this: (*) The program works correctly if it is compiled without optimization, but seg-faults if it is compiled with optimization. This is obviously a copmpiler optimizer bug, but I'm wondering if using LitProg techniques might exacerbate these kinds of bugs. The tangled code from my web source includes a couple of large functions with lots of loops and control structures... A direct consequence of using named sections somewhat in place of function calls. The program works perfectly on some other machines I've tested it on. Has anyone else run into a compiler limit like this one because of using litprog techniques? What did you do?

My second literate program was written in C++ using FWEB, producing a few files and being split across several sources with an attendant generic Makefile that I developed. It also ran perfectly the first time (and I credit that to the care I took when writing it and that the act of writing about what I was doing clarified my own thinking). This post is long enough as it is. I hope to contribue more to this list and I am particularly interested in insights that people have about using Object Oriented Design and Literate Programming.

From: **Kayvan Sylvan**
Date: 14 Apr 1993

The problem I was talking about in my last (long) post, with my first real literate program, was just solved! While I was relaxing, reading my woven output, I spotted a subtle bug that only showed up on the RS6000. I fixed this bug, made some performance improvements while I was at it, and tested the RS6000 build. It worked!!! I'm very happy with the fact that I used Literate Programming, since similar bugs in non-literate programs have stumped me for days, sometimes weeks. Having written the program in an expository mode, I made it easier for myself to understand and debug what I was doing!

From: **Kayvan Sylvan**
Date: 14 Apr 1993

I've probably gone overboard, but what the hell. I grabbed all the WEB tools I could get my hands on. Being between projects does strange things to me, especially when I have "programming fever". I've got CWEB-2.8, FWEB-v130, c-no-web, and noweb installed. Here's my quick thoughts on all these packages:

1. CWEB-2.8 is fairly good for non-ansi C (and Cameron Smith shows a way to use it for ANSI C by his KR-CW-sample). Fairly solid. Nice features. It's not easily configurable, and works easily only with TeX. It's a medium sized package.

2. FWEB-v130 is my current favorite. It handles C, C++, TeX, and a bunch of outdated programming languages that I almost don't believe people are still using. :-) It's nicely configurable. Among its features are a *large* user manual. Among its drawbacks is a *large* user manual. :-) Works with TeX or LaTeX. Large package. Lots of nice features. I've used it the most.

3. noweb. I like its simplicity. It's small and elegant. I'll probably use it for shell scripts, perl scrips, Makefiles, and other applications where literate programming would be good and I have no need for indexes or prettyprinting. It's a nice tool. Works with TeX or LaTeX (probably could change it to work with other formatters too).

4. C-no-web. Cute idea. I'm not sure if I will ever use it. I don't see it as "Literate Programming" since one of the basic ideas of litprog is the ability to organize your input code in any order you want (not tied to the order required by the compiler). I could see using c-no-web to enhance documentation of existing non-literate programs, though.

From: **Osman Buyukisik**
Date: 15 Apr 1993

CLIP: language independent, multiple files may be produced from one input, The WEB part looks like comment in the language you are using, order not important, you need to use your favorite Word processor/typesetter.

FunnelWeb: language independent, same as above but hooked into TeX some people have a working LaTeX version. sizable manual and programmers' manual.

NUWEB: same as above but works with LaTeX. Couple of pages of manual. Program itself written in NUWEB. You get indices for the macros and files produced (just like in FWEB).

All of the above do not pretty print your code and do not produce an index for the words used (other real WEB's do!). They are language independent. If you want code to be pretty printed and use different languages than (c,c++,fortran, TeX) the only option I know is to use SPIDER and produce your own WEB. I personally use NUWEB nowdays for everything. It is a small but powerful program.

From: **Marcus Speh**
Date: 15 Apr 1993

My second literate program was written in C++ using FWEB, producing a few files and being split across several sources with an attendant generic Makefile that I developed.

Did you solve the problem (outlined in the FWEB-FAQ in Question [29]) of presenting a _nice_ table of contents and a combined index which distinguishes between entries from different source files? The example I have in mind is Cameron Smith's hack of cwebmac.sty for the CWEB-KR-sample files. I know that John Krommes has put it in his long list of future enhancements (but it isnt contained in v1.30 as far as I can see).

On Kayvan's list of tools: should merge with David's general FAQ (still in preparation). I only know CWEB (v3.0-beta, which I have found to be faultless so far) and FWEB (v1.30 is beta but John keeps eliminating bugs and it works fine for me). I agree with his opinions. I have CWEB, FWEB, noweb documents on the LitProg page for the World Wide Web. Somebody volunteers to write a few lines (hypertext) for C-No-Web along what I put up for the other LitProg tools? This mainly means one or more links to first-hand information and possibly some preprocessed examples. BTW, I plan to update the FAQ for FWEB v1.30 not later than next month.

From: **Kayvan Sylvan**
Date: 03 May 1993

Did you solve the problem (outlined in the FWEB-FAQ in Question [29]) of presenting a _nice_ table of contents and a combined index which distinguishes between entries from different source files?

No. I started to look into it, but real work priorities intruded. I'm now looking for my next consulting gig, so I'm either going to be real busy or NOT. 1/2 :-)

The example I have in mind is Cameron Smith's hack of cwebmac.sty for the CWEB-KR-sample files. I know that John Krommes has put it in his long list of future enhancements (but it isnt contained in v1.30 as far as I can see).

No, it's not in FWEB yet. I have converted the CWEB-KR-sample to FWEB and doing the combined table of contents and index is on my list of things to do (though I don't want to duplicate any work that John Krommes has already put into this).

I have CWEB, FWEB, noweb documents on the LitProg page for the World Wide Web. Somebody volunteers to write a few lines

(hypertext) for C-No-Web along what I put up for the other LitProg tools? This mainly means one or more links to first-hand information and possibly some preprocessed examples.

Along these lines, and mostly for fun (though I use it all the time now), I developed a generalized Makefile for FWEB/CWEB using noweb!!! It allows me to write Makefile fragments (reminiscent of imake and its "Imakefile") that produce the real Makefile for a WEB-based application. I call it MetaMake. If people are interested, I can send it to them (or to the list).

Current view of literate programming

From: **Vince Mehringer**
Date: 14 Apr 1993

Here's where I stand in relation to literate programming at the moment. Set me straight if I'm wide of the mark on something. In any case I'd like to hear other opinions on the subject. Background: Software developer ~10 years. Mostly C and C++, though some FORTRAN early on and some Lisp recently. Last project involved more than 40 people and resulted in more than 2 million lines of C, C++, and Lisp. Bought _Literate Programming_ when it first came out and have ruminated on the subject since then.

Problems Suitable for LitProg: Navigation. One thing I've seen on project after project is that it is difficult to focus on the appropriate level of the code. The indexing and cross-referencing I've seen in literate programming examples is very encouraging. But one must be able to change contexts _fast_! Double-clicking on the reference and having it on the screen is the kind of speed that is needed. Documentation. If the code and text are in separate files, forget it. And the farther away they are from each other in the same file, the less likely they will stay in synch. I know literate programming doesn't guarantee code and documentation synchronization, but at least it seems to present small chunks of both at the same time. This "in your face" presentation of the documentation seems the best approach so far.

Difficulties with Current Tools: Formatting Language. I'm a software developer -- not a typesetter, and for now I refuse to get sucked into the black hole that is document layout. We have developed coding style guidelines to avoid questions of how far to indent, what declarations should look like, etc. These questions can be answered once, and then productivity is increased. I should _not_ have to deal with learning and coding in a formatting language. This is not a knock against TeX or any other formatting language -- they are extremely useful in the appropriate domain. But to a software developer any formatting language (if one is used) should be completely transparent. Debugging. Ideally this would be on the same document that the program was written. At the very least it should be on unaltered code. I haven't tried Cnoweb, FWEB, or many of the other tools mentioned here, but I am encouraged to read that they transfer code for compilation without making changes. Illustrations. Should be included in a document without difficulty. Right now generating illustrations for a document is difficult enough. Merging them into a literate program is another challenge.

Bottom Line: The hypertext navigation ability rings 100% true with me. Does a tool exist for moving gracefully through the code? The problem of code and documentation getting out of synch gnaws away, but some improvements have been made. But the lack of a tool which hides the formatting language is a real killer. For now I will encourage the evolution of literate programming, dream about what an ideal tool should do, and keep looking for a tool which comes close to the ideal. Oh, and continue developing software as effectively as possible.

From: **Stephen Fulling**
Date: 14 Apr 1993

Vince Mehringer writes: Difficulties with Current Tools: Formatting Language. I'm a software developer -- not a typesetter, and for now I refuse to get sucked into the black hole that is document layout. We have developed coding style guidelines to avoid questions of how far to indent, what declarations should look like, etc. These questions can be answered once, and then productivity is increased. I should _not_ have to deal with learning and coding in a formatting language. This is not a knock against TeX or any other formatting language -- they are extremely useful in the appropriate domain. But to a software developer any formatting language (if one is used) should be completely transparent.

Fair enough, but let's not confuse two very different things. GLOBAL formatting -- the layout of paragraphs and larger units -- can indeed be coded once and for all into macro files, and surely should be in any organization where many people have to work on the same document (or program). Same for coding style.

However, the LOCAL TeX conventions for representing typeset mathematics in ASCII are too valuable in literate programming to give up. The ability to produce readable formulas in the documentation is one of the principal advantages of WEB, and the basic symbols and control sequences needed to do a minimal job of that are presumably as easy to learn as the codes for any word processor that does an even remotely comparable job with mathematics. TeX users quickly get into the habit of using TeX notation in their e-mail, for direct mathematical communication with another human being through an ASCII medium that will never be typeset; TeX is now so

nearly universal in academic math and physics that most people don't even bother to inquire whether the mail recipient is literate in the language. I think that this TeX coding will soon be learned by every calculus student.

Of course, one can't produce perfect typeset output without a deeper knowledge of TeX. However, most people don't worry about polishing until the final draft, after all matters of substance have been settled; in the corporate context this task will presumably be handled by a specialist. Anal compulsives like me who insist on getting every thinspace right in the first draft will have no objection to learning TeX in depth.

From: **Kayvan Sylvan**
Date: 14 Apr 1993

Vince Mehringer writes: Navigation [...]

I use the GNU Emacs based web-mode, which allows me to do outlining (hiding various levels of the input source), jumping to sections and modules, use the web-generated index to jump to uses of identifiers or modules, and a lot more.

Documentation [...]

This is one of the *big* plusses of LitProg for me. My recent experiences have made me totally convinced of LitProg's utility to me *now* (without all the cool Hypertext/GUI advances that are probably in the wings).

Difficulties with Current Tools: Formatting Language [...]

Not all LitProg implementations use TeX. noweb and FWEB both use LaTeX (which stands for Lazy-TeX, I think :-) You don't need to use all the features of TeX to get beautifully typeset programs out of a good LitProg tool.

I'm a software developer -- not a typesetter,

Capt'n, I can't change the laws of software development! :-) But seriously, I'm not a typesetter, or a student, or a researcher, or a professor. :-) I'm a software developer working in the real world, developing systems...

and for now I refuse to get sucked into the black hole that is document layout. We have developed coding style guidelines to avoid questions of how far to indent, what declarations should look like, etc. These questions can be answered once, and then productivity is increased. I should _not_ have to deal with learning and coding in a formatting language. This is not a knock against TeX or any other formatting language -- they are extremely useful in the appropriate domain. But to a software developer any formatting language (if one is used) should be completely transparent.

Sounds like you might want to take a serious look at Norman Ramsey's noweb. It is simple and elegant, completely language independant, doesn't reformat your code, and is very easy to change for other formatters (The WEAVE process is a small shell script).

Debugging [...]

noweb, CWEB, and FWEB all provide support for including "#line" directives into the tangled program. FWEB also provides a breakpoint capability which I found useful in doing web-based debugging. Any compiler/debugger worth the bytes it is written on should be able to use these #line directives to jump you to the WEB source.

Illustrations [...]

If you relax your prohibitions against TeX and LaTeX, this is actaully not too hard. There are various ways of including postscript in TeX documents (my company letterhead is generated this way). I agree with you, though: My fantasy would be of a good GUI tool that allows you to paste graphics right into the WEB document as you are writing it.

Bottom Line: The hypertext navigation ability rings 100% true with me. Does a tool exist for moving gracefully through the code? The problem of code and documentation getting out of synch gnaws away, but some improvements have been made. But the lack of a tool which hides the formatting language is a real killer.

The web-mode for GNU Emacs that I spoke about above does most of this. I use it all the time. It's great.

For now I will encourage the evolution of Literate Programming, dream about what an ideal tool should do, and keep looking for a tool which comes close to the ideal. Oh, and continue developing software as effectively as possible.

I'm starting to believe that literate programming provides many benefits *now*, so while I agree with your sentiment, I'm not going to sit by the sidelines and watch it grow... I'm jumping right into the middle of it.

From: **Marcus Speh**
Date: 15 Apr 1993

Thanks both to Vince and Kayvan Sylvan for their nice reports--keep those keynotes coming, folks. I think communicating among LitProgrammers on personal experiences cannot be overestimated.

Vince Mehringer said: The hypertext navigation ability rings 100% true with me. Does a tool exist for moving gracefully through the code?

This topic has not been beaten to death yet, but seems to come up regularly--for the sake of completeness: I was told by Tim Berners-Lee (who founded the World Wide Web initiative) that the www hypertext line mode browser (version 2.0) is capable of "Literate Programming" as well. This is part of his original message:

BTW ... did you know you can now literate program in HTML, by using (the latest)

```
www -to text/x-c xxx.html > xxx.h
```

to reformat the document as C, with everything except PRE sections commented out? I used txt2html.sed to make the original .html, then I used the nextstep editor to format the file nicely. "text/x-c" is an experimental (x-) MIME content type. The "-to" and "-from" options take (from a limited set) a MIME content type value. You need version 2.0 or later. It's not documented yet. See http://info.cern.ch/hypertext/WWW/Library/Implementation/HTFormat.html for an example header file.

I don't have www 2.0, and no time to bring it up now, but maybe someone who's experimenting with HyperText [HTML] wants to give it a try. All the source is (on the Web and) on info.cern.ch in pub/www.

Module decomposition advice

From: **John Nicoll**
Date: 21 Apr 1993

Well, after reading the LitProg mailings for a while now, I got around to getting hold of WEB (FWEB, in fact, for my PC) and trying things in anger. I'm at the stage where I can convert/write a program (in C/C++) in a `modular' form, understand the error messages, and generate output files.

What has so far taken me the longest time to sort out is the `art' of decomposing into modules. The FWEB (and other, as far as I can see) manuals are silent on this issue - perhaps due to its FORTRAN background. I haven't got a feeling yet for how to best arrange things between modules and functions, and would appreciate advice from more seasoned WEBers on this issue. An example of code written in C/C++ might be useful (perhaps in the FAQ - which also seems silent on the `art'). Thanks for any help in advance.

From: **Marcus Speh**
Date: 21 Apr 1993

John Nicoll said: What has so far taken me the longest time to sort out is the `art' of decomposing into modules. The FWEB (and other, as far as I can see) manuals are silent on this issue - perhaps due to its FORTRAN background.

The fact that it's linked to FORTRAN by name is somewhat unfortunate. IMO, the reason why advice of that sort is lacking is because arrangement of functions, modules and such rather seem to be a matter of personal style. I guess you will have to spend some time to find the way of decomposing which suits you best [this statement is restricted below]. I remember an early discussion on this list on "where to put the main program", and another more recent one concerning the question of whether to put more than one @<section name@>= entries into one section/module.

An example of code written in C/C++ might be useful (perhaps in the FAQ - which also seems silent on the `art').

As for the FWEB FAQ, I'll try to improve on that in the upcoming issue (John Krommes [JAK] does the checking for the final draft of the FAQ, and he may have to say something worthwile, too). There is already a remark on "multiple source files", though. This is on JAK's list of future improvements. Some (rather silly) C++ examples are at ftp.desy.de in directory pub/web/fweb -- a longer one is preprocessed and can be accessed on the World Wide Web (see below). I am working on a C++ library which will provide a fairly

good example, too , but it won't be finished before this summer.

There are already very useful examples available as part of the FWEB distribution: The choice of C, C++ or Fortran should not be an issue, IMO. The layout of your literate programs should rather follow general guidelines as sufficiently described in various articles and manuals. These imaginary 'guidelines' aren't just words: look into the source code of the CWEB and FWEB releases. Probably Levy/DEK have put parts of common.w into the User's Manual for this reason. Altogether, in _my_ programs, the decomposition does not change essentially from one language to another. But again, I _do_ have a Fortran background, so maybe I'm missing the point here. And I wouldnt claim to do an 'art', either. Some comments on this last point by people with another background ?

From: **Stephen Fulling**
Date: 21 Apr 1993

John Nicoll writes: What has so far taken me the longest time to sort out is the `art' of decomposing into modules. The FWEB (and other, as far as I can see) manuals are silent on this issue - perhaps due to its FORTRAN background. I haven't got a feeling yet for how to best arrange things between modules and functions, and would appreciate advice from more seasoned WEBers on this issue. An example of code written in C/C++ might be useful (perhaps in the FAQ - which also seems silent on the `art').

Two books that contain discussions of this issue and example code: 1) Knuth's reprint book, "Literate Programming" (published at Stanford but marketed (also) by the University of Chicago Press). 2) Wayne Sewell, "Weaving a Program" (Van Nostrand, available from TeX Users Group). The 3 programs in chapters 4-6 of Knuth's book are worth studying. You may not agree with all of Knuth's tastes and design decisions (I don't), but as a "strict construction of the intent of the founding fathers" they obviously can't be beat. Original references: Computer Journal 27 ('84); Communications of the ACM 29 (May and June '86).

Unfortunately, most of the published model programs are in Pascal. More attested C and C++ models are badly needed. Several sample C programs are distributed with CWEB, and there is another one in the last appendix of Sewell's book. Sewell discusses modularization (specifically, the horrors of failing to modularize an old program that is being converted (literated?)), but it has been several years since I read the book, so I wouldn't presume to recommend particular chapters at this point.

From: **David Kastrup**
Date: 21 Apr 1993

Well, after reading the LitProg mailings for a while now, I got around to getting hold of WEB (FWEB, in fact, for my PC) and trying things in anger. I'm at the stage where I can convert/write a program (in C/C++) in a `modular' form, understand the error messages, and generate output files.

What has so far taken me the longest time to sort out is the `art' of decomposing into modules. The FWEB (and other, as far as I can see) manuals are silent on this issue - perhaps due to its FORTRAN background. I haven't got a feeling yet for how to best arrange things between modules and functions, and would appreciate advice from more seasoned WEBers on this issue. An example of code written in C/C++ might be useful (perhaps in the FAQ - which also seems silent on the `art').

One thing to look at, of course, are other WEBs. TeX, the program is one to mention. Another rule is that a function might be appropriate if we have things done which are pretty capsulated, and when we might possibly call this function several times. If, on the other hand, we have something which is clearly intended to do a specific subjob of something larger, and will not be of use for anything else, a module is better used. In general, modules should keep pretty small. If you need a nice editor to see which brace belongs to which, chances are you had better split your source up a bit sooner. So, as a rule, modules should be plentiful as regarded to functions.

From: **Norman Ramsey**
Date: 23 Apr 1993

In my opinion, one of the biggest problems of the literate programming community has been that many people expend enormous effort on tools, while there is not a good understanding of how to use even very simple tools. When Carla Marceau and I wrote our paper on our experiences building a 10-20 K line system using Spidery WEB, we tried to explain how literate programming had made our proejct a success, and to be prescriptive about how to use literate-programming tools well. The referees quite rightly pointed out that this part of the paper was vague if not actually content-free, and the final version of the paper contains only some weasel words along the lines of "we're really pleased we used literate programming and we're not sure why." noweb was my effort to scrape away all the complexity of existing tools in order to try to learn something more fundamental about literate programming. I'm pleased with the tool, but I haven't learned the things I had hoped to learn from it. The best advice I can give to aspiring literate programmers is still the same advice I give to students writing papers: write, read your work, and rewrite. Peer review is invaluable if you can get it. Anybody else making better progress?

From: **Lee Wittenberg**
Date: 26 Apr 1993

Norman Ramsey writes: The best advice I can give to aspiring literate programmers is still the same advice I give to students writing papers: write, read your work, and rewrite. Peer review is invaluable if you can get it. Anybody else making better progress?

I'll second Norman's advice. We've got a few people here at Tipton Cole + Co. getting started with literate programming (using noweb and WinWordWEB), and we seem to get our best results by writing, rewriting, and passing the programs around for comment. One of the best things about literate programming, I've found, is that it is possible to pass around a (woven) listing of a program and have people actually read it and make suggestions, something I cannot recall ever happening with a regular ol' source listing.

From: **Kayvan Sylvan**
Date: 04 May 1993

John Nicoll writes: What has so far taken me the longest time to sort out is the `art' of decomposing into modules. The FWEB (and other, as far as I can see) manuals are silent on this issue - perhaps due to its FORTRAN background.

Marcus Speh writes: The choice of C, C++ or Fortran should not be an issue, IMO. The layout of your literate programs should rather follow general guidelines as sufficiently described in various articles and manuals. These imaginary 'guidelines' aren't just words: look into the source code of the CWEB and FWEB releases. Probably Levy/DEK have put parts of common.w into the User's Manual for this reason.

I completely agree with Marcus here. Literate Programs should still follow good programming style. You can do good programming in assembler and bad programming in C++, it's not linked to language. The layout of my literate programs follow a mixture of intuition and computer science. I have a sense of what I want to accomplish and I start by writing the documentation chunk that introduces the program and the motivation for the program.

After the introduction of the program, I use outlining and old-fashioned structured programming techniques (with a smattering of object oriented design) to decompose the problem. I write an overview of the program using named modules (to be defined further inside the WEB). As I get to each module, I write the documentation chunk and write the code, and often find that there is an obvious next step that makes the writing and the programming flow.

Just as in my plain C or C++ programs, I define functions when the set of operations is generic enough to be used by other parts of the program. I use named modules when I am trying to decompose a piece of code that is best understood in overview (rather than overwhelm my reader with details). This makes for an easy to write and easy to understand program.

I also find that object oriented design mixes in very well with literate programming (and takes care of the classic OO problem of how you document your classes in a meaningful way). Literate programmers would do well to keep Miller's 7 +/- 2 law in mind as they weave and tangle their webs. As an aside: a side effect of literate programming for me has been that my programs take less time to write because I'm having so much fun writing them!

Altogether, in _my_ programs, the decomposition does not change essentially from one language to another. But again, I _do_ have a Fortran background, so maybe I'm missing the point here. And I wouldnt claim to do an 'art', either. Some comments on this last point by people with another background ?

Well, I come from a mostly C/Unix background and I found your comments to be emminently reasonable. I don't think you're missing the point.

Comments on simplicity

From: **Bart Childs**
Date: 23 Apr 1993

The inquiry from Chris Flatters on `Supressing index entries in FWEB' and the thoughtful response from Norman Ramsey encourage me to add a few more thoughts.

First, I seriously considered answering Chris' inquiry with:

1. Even if they are seriously long (like many pages), the value of this in most cases is too great to want to throw away.
2. If someone else is going to (possibly) maintain your code, then you really want them.
3. If you are using dvips (and most good drivers) it is fairly easy to create scripts to just not print those pages if that is the problem.
4. Can't that be put in a `.h' file that could have been done as a web and then kept for everybody to use?

Then Norman said: "In my opinion, one of the biggest problems of the literate programming community has been that many people expend enormous effort on tools, while there is not a good understanding of how to use even very simple tools ... the final version of the paper contains only some weasel words along the lines of "we're really pleased we used literate programming and we're not sure why."" That is his opinion and I think based on sound reasoning. The postings on the subject `beginner seeks module decomposition advice' are evidence enough.

He also said, "The best advice I can give to aspiring literate programmers is still the same advice I give to students writing papers: write, read your work, and rewrite. Peer review is invaluable if you can get it." Eloquent! A word of warning, leave your feelings at the door. We are seldom ready for others' constructive criticism. It does not help that most we will get to read our work will not understand what literate programming is about and that we are frequently not true to the intent. (Also, remember this in the `new features' below.)

Finally, Kayvan Sulvan's help prompts one more thought. Simplicity! I have read the FWEB manual many times and don't remember that at all. Have we lost sight of what we are after? Norman's primary reason for noweb was simplicity (his words were `scrape away all the complexity of existing tools'). I look at WEBs and see that most rarely use more than a small fraction of the available commands, features, or whatever you want to call the @-things. Yet, we have had a plethora of postings of requests for `new features'.

I have observed that if a document is being prepared and a difficult (formatting) challenge comes along, then there is a real dichotomy: 1) if TeX (or the like) is being used the user may spend inordinate amounts of time trying to `make it perfect.' 2) if it can't be found in two minutes on the WYSIWYG's pull down menu, the mind is quickly set to ``God did not intend such to be done!' Of course the first is from observations during the preparation stage and the second from questioning `why this?' after seeing student papers thought to be perfect because they: have an aligned right margin, have been within a mile of a spell checker, and have a gaudy excess of bold font usage.

It seems to me that some of our problems are due to this `TeX nature.' Don Knuth created some great research problems solving many aspects of TeX's and MetaFont's problems. Quotes from two I think deserve to be quoted: "One way of improving efficiency is by restricting the power of the language" --- Tony Hoare. "The need for more elaborate man/machine interaction can certainly be enhanced by designing more incomprehensible systems" --- Edsger Dijkstra.

Some of the themes that I hope are woven into this commentary are what I feel we used as justification for the design decisions in Mark Motl's web-mode for emacs. We wanted it to be easy to do some of the more difficult things like: 1) Reuse section names (the @< .. @> things) with minimal typing. In web-mode's current form, a list of them are maintained and you select them by use of emacs completitions. I never did like Knuth's use of ellipses although I certainly understand why... 2) Navigation by chapters (major sections), sections, cross reference lists, index, ... It can be considered to be a restricted hypertext mechanism. 3) Creation and reuse of user supplied index entries (@. **@>, @^..@>, and the other one). The outline editing was added later and has been a real winner too.

In much the spirit of Norman's comment, I would really like to get a lot of data back on how people use it before we add a lot more functions. Unless it is defeated, a journal file is written that we get lots of data from. However, most have come from within our local community. Indeed, many users don't use a lot of the functions that I believe would be of great benefit. However, isn't that the usual story? I have seen a large number of people who will move down 100 lines by 100 `down arrow's rather than 5 `page down's!

Constructive criticism is appreciated. Bart Childs

From: **Adrian Clark**
Date: 27 Apr 1993

In much the spirit of Norman's comment, I would really like to get a lot of data back on how people use it before we add a lot more functions.

I have put together Perl versions of most of Norman's excellent noweb system for my own consumption (this was for use on DOS, before someone came along with a DOS implementation of noweb). One of the things my Perl stuff can do is output statistics on the number of times each internal `@' construct is used. This was very helpful in optimizing the order of processing each construct in a else..if chain. Surely it would be pretty simple to do this to any web-like package? This would yield invaluable information on precisely which features were essential, which were highly desirable, and which totally superfluous. Much more reliable than allegory.

For anyone burning to know, @nl is the most common noweb entity, followed by @text. For most of the programs I've looked at, everything else is used less than one-tenth as frequently.

From: **Kayvan Sylvan**
Date: 4 May 1993

[... about web-mode ...] In much the spirit of Norman's comment, I would really like to get a lot of data back on how people use it before we add a lot more functions. Unless it is defeated, a journal file is written that we get lots of data from. However, most have come from within our local community. Indeed, many users don't use a lot of the functions that I believe would be of great benefit. However, isn't that the usual story? I have seen a large number of people who will move down 100 lines by 100 `down arrow's rather than 5 `page down's!

I use web-mode to do outlining, jumping back and forth between sections, reading WEB code as well as writing it, view section names and jump to it, look at a variable in the index and go back and forth between code and index, etc. The most common operation I use is the automatic module name generation and selection mechanism.

Duplicate sections and good webbing style

From: **Lee Wittenberg**
Date: 30 Apr 1993

The recent discussions about the proper use of modules vs. procedures and other points of style got me thinking about what would constitue "good webbing practices." I'm not going to try to generate an "Elements of Webbing Style" here, but a thought struck me about style and duplicate code sections (the ones defined with +\equiv) that I thought might be of general interest: It would seem that the interests of good style would dictate that the code in the different chunks of a duplicated section should be completely independent. For example,

```
@ @<A duplicated section@>=
    x = y + z;

@ @<A duplicated section@>=
    a = b + c;
```

would be okay, but

```
@ @<A duplicated section@>=
    x = y + z;

@ @<A duplicated section@>=
    a = x++;
```

wouldn't. The reason for this is that in the former case, a reorganization of the web is guaranteed not to screw up the program, while in the latter case, there is a very real danger that reorganizing will break the program (and this would be a very difficult bug to find). This has some very interesting implications. If the duplicate sections are independent, then they can be executed in parallel, which leads to a very natural way to express parallelism (by way of literature):

```
@*Chapter n.
In this chapter, some really interesting stuff happens.
@<The stuff that happens in the first chapter@>=
        /* A */

@ Meanwhile, back at the ranch~$\ldots$
@<The stuff that happens in the first chapter@>=
        /* B */
```

A smart tangle could generate the equivalent of cobegin A; B coend Declarations, of course, would not need the cobegin/end, but they (conceptually) execute in parallel already. This technique will, of course, cause severe problems with my second example, but (as I pointed out earlier) it was bound to break eventually, anyway.

From: **Preston Briggs**
Date: 30 Apr 1993

I find that I use "duplicate sections" almost exclusively for accumulating declarations and almost never for code. When I want several pieces of code to contribute to a single module, I organize it hierarchically, and explain the subdivision right away. For example, instead of

```
@<The main code@>=
  @<foo@>

@<The main code@>=
  @<bar@>

@<The main code@>=
  @<quux@>
```

possibly scattered throughout the document, I prefer to write

```
@<The main code@>=
```

```
@<foo@>
@<bar@>
@<quux@>
```

I find the idea of having tangle perform automatic parallelization to be a little scary :-)

From: **David Kastrup**
Date: 01 May 1993

I find that I use duplicate sections for code as well: mostly for cases in switches. There I have no bad feelings about them.

From: **Kayvan Sylvan**
Date: 04 May 1993

Lee Wittenberg makes some interesting points about duplicate code sections. Looking in my code, there are a few places where I use duplicate (appended to) sections:

(*1*) Accumulating declarations. This is especially true in C (since I can put declarations most anywhere in C++ and the need is lessened). A typical example is:

```
    @ In |Print_Postscript_For_Board()|, we look at the Chess position and
    print the board using the Foresythe notation as implemented in
    Walker's chess font.

    @<Printing Routines@>=
    void Print_Postscript_For_Board( BOARD *my_board )
    {
            @<Variables local to |Print_Postscript_For_Board()|@>@;

            @<Prepare the postscript code@>@;
            @<Output code to user-specified place, default |stdin|@>@;
    }
```

And then I can do something like this:

```
    @ To prepare the postscript code, we need to look at the structure
    elements in each cell of |my_board|. We use the |score| and
    |whose_move| variables in the code that follows.

    @<Variables local to |Print_Postscript_For_Board()|@>=
    int score;  /* this is part of the chess game listing */
    enum sides { WHITE, BLACK } whose_move;

    @ @<Prepare the postscript code@>=
    score = my_board[0].score;  /* the score for this snapshot */
    if (my_board[0].is_white)
            whose_move = WHITE;
    else
            whose_move = BLACK;
```

(*2*) Accumulating related functions. This mechanism makes it easy for me to have *one* well-structured un-named section, rather than lots of unnamed sections sprinkled throughout my code. Example:

```
@ The foobarbletch program needs to do many somewhat unrelated
things. Here is the overview of this program.

    @A @% FWEB un-named section (equivalent to @c in CWEB)
    @<Header Files@>@/
    @<Global Variables@>@/
    @<Chess Game Parsing Routines@>@/
    @<Postscript Output Routines@>@/
    @<The main program@>@/
    @<Utility Routines@>@/
```

And then @<Utility Routines@> becomes the catch-all bucket for any useful function that I need to use elsewhere in the program.

(*3*) Un-tangling large switch statements. Various parsers and simple state machines are best explained on a "case" by "case" basis.

```
    @ So now, we are ready to read and decompose our input file, producing
    the intermediate format that is easy for us to generate queries from.

    @<Parse input, producing balanced AVL tree@>=
    @<Read |token|, using lex@>@;
    switch(token)
    {
            @<Appropriate actions based on value of |token|@>
```

```
default:
        /* We shouldn't get here!!! Ever!!! */
        fprintf(stderr, "I'm DYING! I don't know about token: <%s>\n",
                token.string);
        abort();    /* maybe a core dump will do some good */
}

@ If we are handed a |QUERY| type, we need to perform the action that is
specified by the |type| flag of |token|.

@<Appropriate actions based on value of |token|@>=
case QUERY:
        run_actions(token, token.type);
        break;

@ If instead, we have an integer, push it onto our value stack so it
can be used as an operand for later actions.

@<Appropriate actions based on value of |token|@>=
case INTEGER:
        i = atoi(token.string); /* better add integer type to the lexer */
        push_to_value_stack(i);
        break;
```

That's pretty much it, I think. In general, I find that the pieces of the named (multiply defined) section should be as modular and as self-contained as possible. In the the switch statement, each case block can be an independant, self-contained unit, and rearranging the web should not cause it to break.

Anybody there?

From: **Kayvan Sylvan**
Date: 13 May 1993

Is litprog just being quiet? What are other literate programmers doing out there?

From: **Sean Boyle**
Date: 13 May 1993

Being a plebe to literate programming, I am still fiddling around. I think I've figured out that I like FunnelWeb as I can use it for any arbitrary language and it doesn't muck with it too bad. I'm not fond of losing my tab key, sigh... Is there a review of the various systems out there somewhere?

I'd certainly like to see some good examples of literate programming so that I can establish a discipline. I've seen the bibliography listing and it looks rather extensive. It would be nice if some of the stuff was online where I could get at it. I suppose it is all new enough that there aren't a lot of hard and fast rules, but there must be some scars that others could show me to save me from some grief. I appreciate the comments on style which the FunnelWeb user's guide points out, but I'd like more.

I suppose that the next battle I'm up against is a manager who believes that all staff members should use Frame (I think it's supposed to be another productivity tool...) so that we can share documents. Unfortunately, this system doesn't fit very well into that paradigm. What is worse, my position of having a tool which produces a document which in turn makes a code review a breeze is somehow untenable.

From: **Mary Bos**
Date: 13 May 1993

Sean Boyle writes: I suppose that the next battle I'm up against is a manager who believes that all staff members should use Frame (I think it's supposed to be another productivity tool...) so that we can share documents. Unfortunately, this system doesn't fit very well into that paradigm. What is worse, my position of having a tool which produces a document which in turn makes a code review a breeze is somehow untenable.

I, too, work in the land of FrameMaker (and associated products) users. I've heard rumour Frame technologies is working on a conversion package to take TeX files, but no dates or anything definite.

From: **Kayvan Sylvan**
Date: 13 May 1993

Sean Boyle writes: Being a plebe to literate programming, I am still fiddling around. I think I've figured out that I like FunnelWeb as I can use it for any arbitrary language and it doesn't muck with it too bad. I'm not fond of losing my tab key, sigh...

What does FunnelWeb do to the sources? Is it like noweb (leaving it all verbatim the way you typed it)? What about indexes or a table of contents?

Is there a review of the various systems out there somewhere?

There have been several informal reviews by Marcus Seph, myself and others. Look in the litprog archives. My present favorites are FWEB for C/C++/TeX, and noweb for everything else (Perl scripts, Makefiles, etc.) With the final of FWEB 1.30 about to be released, I expect an even better FWEB (in terms of robustness and support for multiple files, etc.)

I'd certainly like to see some good examples of literate programming so that I can establish a discipline. I've seen the bibliography listing and it looks rather extensive. It would be nice if some of the stuff was online where I could get at it.

There's Cameron Smith's multi-file CWEB example (KR-CWEB-SAMPLE or something like that) that implements the calculator program from K&R second edition. I'm working on an FWEB sample program in my copious spare time, see my signature line ;-). My sample program will be made available when I finish it. :-)

I suppose it is all new enough that there aren't a lot of hard and fast rules, but there must be some scars that others could show me to save me from some grief. I appreciate the comments on style which the funnelWeb user's guide points out, but I'd like more.

Once again, check the archives. There was a discussion about this recently with the subject: "beginner seeks module decomposition..." If you don't have access to the archives, I can probably dig up the relevant articles and send them to you.

I suppose that the next battle I'm up against is a manager who believes that all staff members should use Frame (I think it's supposed to be another productivity tool...) so that we can share documents.

You may want to look into noweb and see how you can integrate it with Frame. A lot of people would be interested in this (even though Frame is far inferior to TeX in its abilities).

Unfortunately, this system doesn't fit very well into that paradigm. What is worse, my position of having a tool which produces a document which in turn makes a code review a breeze is somehow untenable.

How is it untenable? I've found that code reviews are a pleasure with weaved output, versus being a boring and tedious exercise in self-restraint ;-)

From: **John McArthur**
Date: 13 May 1993

What are other literate programmers doing out there?

You asked, so I will tell you. I work for a Typesetting company. I do a lot of TeX. So I have been working on my personal port of TeX. This now works and passes the Trip test. This is done using TopSpeed Pascal and WEB. To do this job I had to convert Tangle 4.1 and Weave 4.4 to the TopSpeed Pascal. One of the other WEB programs I work with is DVIAPS. This is a propritary program that we bought from ArborText. It takes .dvi files and converts them to .icl files for our Aps typesetters. I have extensively re-written DVIAPS. It now works with Borland Pascal 7.0 using the Dos Extender provided by Borland.

I have also started work on re-writing MANGLE and MEAVE, the Modula 2 versions of Tangle and Weave. Personally, I think Mangle and Meave are totally useless as they currently stand. One of the things I want to do is write a version of Tange/Weave to work with PAL. I do a lot of database programming in Paradox.

From: **Dominique Dumont**
Date: 14 May 1993

I suppose that the next battle I'm up against is a manager who believes that all staff members should use Frame (I think it's supposed

to be another productivity tool...) so that we can share documents. Unfortunately, this system doesn't fit very well into that paradigm. What is worse, my position of having a tool which produces a document which in turn makes a code review a breeze is somehow untenable.

Yep, I think I'm going to have the same problem. Most people think that if it's wysiwig it's better. But that heavily depends on the way you think and on the way you work. But now, I'm preaching people who are already converted. BTW, if you want to know what I'm currently doing , I've quit hardware development and now I'm learning all I need to know about software (C++, smart methodologies [zut, how can I fit web in these ?], AIN, TMN ...) Wish me luck I'm going to be quite busy the next few months.

From: **Anthony Coates**
Date: 14 May 1993

I don't know if I have mentioned what I use FunnelWeb for, but I might anyway, since it seems a little different from what most do. My aim is to create 'all-in-1' papers, where imbedded in the text for the paper is the code which generates the results. So far, this is little different to what anyone else does. The main difference is that I then want the results of the programs inserted into the paper. To achieve this, a common route is

- FunnelWeb generates a Maple file for the calculations and a '.ptx' file for LaTeX without results (.ptx is my own choice, signifying pre-TeX).
- The Maple file is run and generate a small Maple-loadable library of results.
- The .ptx file is then processed by Dougal Scott's texproc program, which he kindly sent to me. This program searches a TeX file for a set of commands bracketted by special comment lines, runs the text through the selected program, and then replaces the text with the result of the program. For me, for each set of results, Maple is run and the library of results loaded. At the appropriate point in the file, the appropriate results are automatically converted by Maple into LaTeX by some routines I wrote (I didn't like the LaTeX code generated by the 'latex' function supplied with LaTeX).
- Finally, I end up with a LaTeX file with all the source code and results together.

Perhaps the main area in which the current languages don't support such a process is it is assumed that the TeX code with the documentation is to remain immutable, and so, in FunnelWeb at least, one can't use macros within the TeX code itself, only within the code sections. For my application, this is sometimes a nuisance, because I want macros, but the TeX ones won't work until too late in the process, and the FunnelWeb ones won't allow me to affect the text. So any change in macros to change the document involves two sets of macros, one for FunnelWeb, and one for LaTeX. That said, I am very happy with FunnelWeb, and I can hardly complain at my using it for something that it wasn't really designed for.

From: **Mary Bos**
Date: 14 May 1993

I suppose that the next battle I'm up against is a manager who believes that all staff members should use Frame (I think it's supposed to be another productivity tool...) so that we can share documents. Unfortunately, this system doesn't fit very well into that paradigm. What is worse, my position of having a tool which produces a document which in turn makes a code review a breeze is somehow untenable.

Dominque Dumont writes: Yep, I think I'm going to have the same problem. Most people think that if it's wysiwig it's better. But that heavily depends on the way you think and on the way you work. But now, I'm preaching people who are already converted.

Just present a seminar on using literate programming to other fellow software engineering students. Most of my peers were hostile to literate programming and didn't like the idea of code reviews using weave output. Most of the reasons revolved around not seeing the code directly, WYSIWIG, and the idea of writing english and not compiler constructs. Now, if we are the current crop of SE's and the future (?) crop of managers - literate programming still has a lot of persuading to do. My peers might listen if the cost/benefits for literate programming (and WEB) were documented versus the cost/benefits for the standard code construction (and documentation). The positive side was WEB fits in well with the Object Oriented Programming packaging on non-objected oriented compilers.

From: **Trevor Jenkins**
Date: 14 May 1993

Is litprog just being quiet?

Yes, sadly. :-(

What are other literate programmers doing out there?

Trying to drum up more support by exchanging long mail messages with at least one other UK netter interested in literate programming. I'm also trying to screw up the courage and ask "Has anyone got a version of WEB that accepts VAX BASIC?" I'd rather use C/CWEB but the customer (and the recalcitrant boss) have mandated VAX BASIC. :-| Oh well I've asked it anyway. :-)

From: **Lee Wittenberg**
Date: 14 May 1993

Sean Boyle writes: Being a plebe to literate programming, I am still fiddling around. I think I've figured out that I like funnelWeb as I can use it for any arbitrary language and it doesn't muck with it too bad. I'm not fond of losing my tab key, sigh...

You might want to try Noweb. It's also language-independent, and in addition, it allows you to specify tab stops (somewhat) with the -t option.

Is there a review of the various systems out there somewhere?

Dunno.

*I'd certainly like to see some good examples of literate programming so that I can establish a discipline. I've seen the bibliography listing and it looks rather extensive. It would be nice if some of the stuff was online where I could get at it. I suppose it is all new enough that there aren't a lot of hard and fast rules, but there must be some scars that others *could* show me to save me from some grief. I appreciate the comments on style which the funnelWeb user's guide points out, but I'd like more.*

Actually, one of the things I like about literate programming is finding my own style guidelines. I've found that I usually write a first draft pretty much as "stream of consciousness." Once I get it working, I look at how lousy the structure is, and redo it (which isn't as hard as it sounds, since most of the code chunks remain the same; I only reorganize the explanations). I think that recognizing good programming (or webbing) style requires more reading than writing of programs. In particular, I think this comes from reading other peoples' programs (to which one has no emotional attachment), and critiquing (sp?) them. Some people can do this for their own programs, but it's always easier to criticize someone else's work.

I would be willing to compile an "Elements of Good Webbing Style" list, and "publish" it via this discussion group, if people are willing to send me (direct) lists of rules they've found useful. Anyone who has such rules, please condense them to a short, simple statement, and send them to me, along with a brief explanation of how the rule is useful. I'll compile them and post them to the list in a week or two. Also, let me know if you wish your contribution to be anonymous, as I intend on crediting each contribution to its original author (if you wish to be known as a contributor to the list, but don't wish your name to be directly linked to your rules, we can do that as well).

I suppose that the next battle I'm up against is a manager who believes that all staff members should use Frame (I think it's supposed to be another productivity tool...) so that we can share documents. Unfortunately, this system doesn't fit very well into that paradigm. What is worse, my position of having a tool which produces a document which in turn makes a code review a breeze is somehow untenable.

I've got the same problem here (at Tipton Cole + Co., not UT Austin). Two of us are producing lovely code using Noweb, but our boss and another of our programmers are confirmed WYSIWYGers and can't be bothered learning TeX. The other programmers are interested in literate programming, but are willing to go either way. The boss is willing to do literate programming if he can use Word for Windows, but is unwilling to see the advantages of Noweb over WinWordWEB (automatic cross-referencing, RCS/PVCS version control, freedom from Microsoft's whims, etc.).

I seem to recall someone mentioning that they were working on a WEB system for Framemaker (or is that different than Frame -- it's hard to keep things straight these days). You may have to end up doing what I did with Word for Windows -- developing your own Frame-based WEB system. WinWordWEB is primitive (Word's macro language doesn't permit more), but it does the job, and it's better than no webbing at all. A show of hands please: How many of you out there have gotten so spoiled by literate programming that you'll fight to the death rather than program without your literate programming tools? [Notice that my hand is raised.]

From: **Lee Wittenberg**
Date: 14 May 1993

Jeffrey McArthur writes: One of the things I want to do is write a version of Tange/Weave to work with PAL. I do a lot of database programming in Paradox.

I spent a couple of months last year trying to use Spidery WEB to generate a WEB system for PAL. I gave it up when I discovered how incredibly convoluted PAL syntax is. Some keywords are reserved, some aren't, and *so much* depends on context. Spaces are significant in some places but not in others, and it usually takes a bit of semantic (in addition to syntactic) analysis to determine the appropriate situation. However, we do a lot of PAL programming here, and I am loath to give up literate programming merely because of an intractable language (see my previous reply to Sean Boyle's note). I use Noweb for all my PAL (and ObjectPAL) programming. It works beautifully (although the code is typeset as prettily as CWEB, FWEB, Spidery WEB, etc.). In fact, my boss is taking a notebook entitled "Literate Programming in PAL and ObjectPAL" (containing about half a dozen Noweb PAL programs) to the Borland Conference in San Diego next week in order to show around (and incite as much interest as possible in literate programming techniques). I am also trying to get the Paradox Informant to publish one of my programs as an article. Anyway, I know of 3 people (including myself) who are doing literate programming in PAL. Two of us are using Noweb, and the third is using WinWordWEB (and has yet to complete his first literate program, although I will admit what he has so far looks pretty impressive).

I should also mention that, as far as I can tell, FunnelWEB is also a worthwhile choice for PAL programming, although I've never actually used it. Noweb works perfectly for PAL, plus it allows me to use LaTeX (I think FunnelWEB is restricted plain TeX, but I could be wrong). Noweb is available via anonymous ftp from bellcore.com (I forget the directory). I don't know where FunnelWEB is accessible from, but I'm sure someone else on the net will know.

From: **Anthony Coates**
Date: 15 May 1993

Lee Wittenberg writes: I should also mention that, as far as I can tell, FunnelWEB is also a worthwhile choice for PAL programming, although I've never actually used it. Noweb works perfectly for PAL, plus it allows me to use LaTeX (I think FunnelWEB is restricted plain TeX, but I could be wrong). Noweb is available via anonymous ftp from bellcore.com (I forget the directory). I don't know where FunnelWEB is accessible from, but I'm sure someone else on the net will know.

A few folks have commented about FunnelWeb and LaTeX. As a FunnelWeb user, let me point out that FunnelWeb was constructed with the idea of being independent of the choice of formatting language, and so has it's own concept of sections, subsections, etc. Despite that, to date the only formatter it supports is TeX. However, it is possible to tell FunnelWeb (as I do) to forget about it's own formatting conventions and just treat the text as a TeX file. This is my preference; I find LaTeX's own sectioning comments fine for what I want. Yes, LaTeX, because once you tell FunnelWeb that you are writing in TeX, you can write in LaTeX and just process the resulting text file in LaTeX. Other that a slight sensitivity about needing to place FunnelWeb macros after the \begin{document} line (I use the LaTeX 'article' style, but this is not at all obligatory), there are no problems at all.

For those who would rather use FunnelWeb's own sectioning commands with LaTeX, I remember that in fact someone did write the appropriate code to support this, but I can't remember who now (I don't use these sectioning commands, so I don't use this code); can someone remind us who did the conversion, and where it is? In any case, I don't myself think that there is any bar to stop LaTeX users working with FunnelWeb, though I would be interested in any opinions to the contrary.

From: **Humberto Zuazaga**
Date: 15 May 1993

Mary Bos writes: I just present a seminar on using literate programming to other fellow software engineering students. Most of my peers were hostile to literate programming and didn't like the idea of code reviews using weave output. Most of the reasons revolved around not seeing the code directly, WYSIWIG, and the idea of writing english and not compiler constructs. Now, if we are the current crop of SE's and the future (?) crop of managers - literate programming still has a lot of persuading to do. My peers might listen if the cost/benefits for literate programming (and WEB) were documented versus the cost/benefits for the standard code construction (and documentation).

I think thay might have a point. Woven code that uses \lnot and \neq in the code takes some getting used to. I think language constructs should not be translated (or optionally translated). In addition, it takes a little time to get used to the out of order presentation of the code, but with a little work on document structure, and by presenting related sections close together the ordering can be a boon.

A poorly woven web is worse than a mediocre non-literate program, and I was very poor at weaveing for a long time (and I already was good at programming). The added complexity does make for less understandable programs until the idiom is mastered. (I still think I need some work at web structure.)

From: **Preston Briggs**
Date: 15 May 1993

I just present a seminar on using literate programming to other fellow software engineering students. Most of my peers were hostile to literate programming and didn't like the idea of code reviews using weave output.

This is all wrong. We've been doing walkthroughs with poorly-documented C, heavily documented C, and woven cweb. The CWEB was far and away the best. The indices and crossreferences are worth my weight in gold. Get 'em to actually try it try a walkthrough.

If we are the current crop of software engineers and the future (?) crop of managers - literate programming still has a lot of persuading to do.

This is fairly depressing. If they worked for me, they'd be last week's crop of SE's. Not so much because they don't obey my every whim, but because they don't recognize good ideas when they see them.

From: **Lee Wittenberg**
Date: 17 May 1993

Trevor Jenkins asks: I'm also trying to screw up the courage and ask "Has anyone got a version of WEB that accepts VAX BASIC?" I'd rather use C/CWEB but the customer (and the recalcitrant boss) have mandated VAX BASIC. :-| Oh well I've asked it anyway. :-)

You might want to try Noweb, or FunnelWEB, or one of the other language- independent literate programming systems. If you want beautifully typeset code, you might want to build your own BasicWEAVE/TANGLE with Spidery WEB. A language like Basic, with a fairly simple syntax, should be relatively easy to generate a Spidery grammar for. Noweb is available via anonymous ftp from bellcore.com (I forget the directory), Spidery WEB from pip.shsu.edu (tex-archive/web/spiderweb). I'm afraid I don't know where FunnelWEB is available.

P.S. Noweb has a feature that is nice if your boss (and customer) decide that you have to "give up this literate programming nonsense." It comes with a tool called "nountangle" that tangles your code into a program (as the regular notangle does), but includes your internal documentation as program comments in the appropriate places. This allows you to experiment with literate programming, and in Norman Ramsey's words, "If the experiment is unsatisfying, it is easy to abandon, because notangle's output, unlike TANGLE's is readable." Personally, I'd try building a Spidery WEB first. I'm partial to the better typesetting of code in the language-dependent WEB systems, but that's a matter of personal taste.

From: **Lee Wittenberg**
Date: 17 May 1993

I just present a seminar on using literate programming to other fellow software engineering students. Most of my peers were hostile to literate programming and didn't like the idea of code reviews using weave output. Most of the reasons revolved around not seeing the code directly, WYSIWIG, and the idea of writing english and not compiler constructs. Now, if we are the current crop of SE's and the future (?) crop of managers - literate programming still has a lot of persuading to do. My peers might listen if the cost/benefits for literate programming (and WEB) were documented versus the cost/benefits for the standard code construction (and documentation).

Humberto Zuazaga writes: I think thay might have a point. Woven code that uses \lnot and \neq in the code takes some getting used to.

I agree with Humberto's observation, but think it's interesting that \lnot and \neq (and \land, and \lor, etc.) should "take some getting used to." After all, the corresponding symbols NOT/!, <>/!=, AND/&&/.AND., etc. were chosen simply because early card and tape punches did not provide the necessary standard mathematical symbols! Take a look at the Algol-60 Revised Report, with its hardware, reference, and publication languages, and notice that the publication language allows things like Greek letters as identifiers and raised exponents. Note also that the REFERENCE language used the standard mathematical symbols for \neq, \lor, \times (!), etc. It was only in the hardware language that concessions were made to the machine.

We've gotten into the habit of designing our languages around machine needs rather than human needs. I see literate programming as a way of going back to the latter (I say "going back" primarily because of Algol-60). Let's start designing new programming languages with an eye for human readability, using standard symbols that have been around for hundreds (sometimes thousands) of years (e.g. \times and \neq) rather than those that have been around for less than 50 (e.g. * and <>). [End of Tirade]

From: **Lee Wittenberg**
Date: 17 May 1993

If we are the current crop of software engineers and the future crop of managers - literate programming still has a lot of persuading to do.

Preston Briggs writes: This is fairly depressing. If they worked for me, they'd be last week's crop of software engineers. Not so much because they don't obey my every whim, but because they don't recognize good ideas when they see them.

Unfortunately, most bosses are not as open as Preston (and the guy I'm working for this year while on sabbatical). It seems to come down to my father's theory of middle management: "It's easier for a middle manager to say `no,' because that way he (or she) can't get into trouble with the higher-ups. Saying `yes' involves sticking one's neck out."

From: **Don Grodecki**
Date: 18 May 1993

If we are the current crop of software engineers and the future (?) crop of managers - literate programming still has a lot of persuading to do.

Preston Briggs writes: This is fairly depressing. If they worked for me, they'd be last week's crop of SE's. Not so much because they don't obey my every whim, but because they don't recognize good ideas when they see them.

Unfortunately, most bosses are not as open as Preston (and the guy I'm working for this year while on sabbatical). It seems to come down to my father's theory of middle management: "It's easier for a middle manager to say `no,' because that way he (or she) can't get into trouble with the higher-ups. Saying `yes' involves sticking one's neck out."

This is probably why the current theories are to "flatten" middle management out of the structure. (PS: I am a middle manager who loves LitProg)

From: **Kayvan Sylvan**
Date: 17 May 1993

Lee Wittenberg writes: A show of hands please: How many of you out there have gotten so spoiled by literate programming that you'll fight to the death rather than program without your literate programming tools? [Notice that my hand is raised.]

My hand is raised too.

From: **Eric van Ammers**
Date: 26 May 1993

Lee Wittenberg writes: A show of hands please: How many of you out there have gotten so spoiled by literate programming that you'll fight to the death rather than program without your literate programming tools? [Notice that my hand is raised.]

Although a bit late, I certainly would like to raise my hand. I have been using the basic technique for many years and for me it is the most convincing presentation of a program. The big problem I face, and all of us literate programming-ers expereince the same, is that we cannot bring to words where exactly is the power of the literate programming paradigm. I have several times before argued that we should discuss this sort of "philosophical" issues on the net. Once again I ask to all of you: Can we somehow make explicite the power of literate programming. Maybe this will finally convince managers too.

From: **Kayvan Sylvan**
Date: 05 Jun 1993

Lee Wittenberg writes: A show of hands please: How many of you out there have gotten so spoiled by literate programming that you'll fight to the death rather than program without your literate programming tools? [Notice that my hand is raised.]

Eric van Ammers writes: Although a bit late, I certainly would like to raise my hand. I have been using the basic technique for many years and for me it is the most convincing presentation of a program.

Completely agreed. I've only been using it for some months now, but I was introduced to the conecpts of litprog years ago.

The big problem I face, and all of us literate programming-ers expereince the same, is that we cannot bring to words where exactly is the power of the literate programming paradigm. I have several times before argued that we should discuss thIs sort of "phylosophical" issues on the net. Once again I ask to all of you: Can we somehow make explicite the power of literate programming. Maybe this will finally convince managers too.

Okay. Here's a first shot at an verbalizing the power of literate programming. First of all, literate programming is unabashedly fun! I

know, I know, this is not the type of thing that would sway a manager, but I have to be honest first before getting to the objective measures. As I write a literate program, I focus on the essentials of the program at the higher levels of abstraction (without needing to deal with the lower level details). This makes for a program that flows nicely from high level overview to low level implementation details. The resulting program is also easy to write and easy to read (for someone else, or for myself, months later).

Most important of all, I'm not limited to the language's requirements for program order or its documentation mechanisms. Literate Programming allows me to document my program in small pieces, and to present those pieces in whatever way makes the most sense for myself and my reader. As in writing a story or a paper, being conscious of my audience forces me to write better programs.

In terms of the process, I find literate programs, once you've learned the methodology, take slightly more time to craft than non-literate ones do. This is because you take more care with the pieces and you are also writing documentation that goes along with the pieces. The debugging time for literate programs, in my experience, is orders of magnitude less than for non-literate ones. My literate programs work reliably on at most the third or fourth iteration. Well-designed non-literate programs are still much harder to debug and test, in my limited experience. This is the real power of literate programming in my business.

In short: Literate programming is fun. Literate programs are easier to compose (since I can pseudo-code my way to completion in a stream of consciousness fashion) than non-literate ones. Literate programs require almost no debugging and many times will work the first time. The products of literate programming are much better documented (and thus much more maintainable) programs. There's some investment of time in learning the tools, but the investment pays off quite quickly in improved code quality and maintainability.

Unresolved sections in CWEB

From: **Zdenek Wagner**
Date: 27 May 1993

I am writing a C++ pure virtual class in CWEB. The class is intended to be used by other programmers even those who do not know WEB (they will just get the tangled code and printed documentation) and also those who are not very familiar with object oriented programming. Therefore I should supply good information how they will derive their own classes.

And now comes the question. The WEB file should contain examples which should be formatted in the similar way as the C-part. I do not like to study the macros from "cwebmac.tex" and do the formatting by hand. I therefore need some trick to put blocks which will be ignored by tangle and formatted like C by weave. The trick with @(nul@>=... will not work because I intend to leave some modules unresolved. Imagine the part of the example:

```
if (errorCode != 0) {
 @<Display error message@>@;
}
```

I will not specify how to display the error message because each programmer has his own way for managing error messages or even his own libraries. It is even more evident on data input and output of results which does not depend upon the properties of the base class but on the nature of the problem. Is there a solution in CWEB? Thanks in advance for your answers.

From: **Francisco Figueirido**
Date: 27 May 1993

Zdenek Wagner writes: And now comes the question. The WEB file should contain examples which should be formatted in the similar way as the C-part. I do not like to study the macros from "cwebmac.tex" and dothe formatting by hand. I therefore need some trick to put blocks which will be ignored by tangle and formatted like C by weave.

I use FWEB rather than CWEB but I would suggest sending the C code to a separate file (something like dustbin.c). This way tangle will happily throw them away while weave will format them correctly (this works in FWEB).

From: **Kayvan Sylvan**
Date: 27 May 1993

Zdenek Wagner writes: And now comes the question. The WEB file should contain examples which should be formatted in the similar way as the C-part. I do not like to study the macros from "cwebmac.tex" and do the formatting by hand. I therefore need some trick to put blocks which will be ignored by tangle and formatted like C by weave. [...]

The way that comes to my mind is judicious use of #ifdef directives in the C part of your code. This will make blocks that are tangled, but ignored by your C compiler.

[...] I intend to leave some modules unresolved. Imagine the part of the example:

```
if (errorCode != 0) {
  @<Display error message@>@;
}
```

I will not specify how to display the error message because each programmer has his own way for managing error messages or even his own libraries. It is even more evident on data inputand output of results which does not depend upon the properties of the base class but on the nature of the problem.

This would seem to imply that you're attempting to get these other people to write WEB code that interfaces with yours. Are you? Or are they going to be writing C++ code that uses the tangled output? I think getting other people to write in WEB in small doses (and getting them addicted!) is a good thing. :-) However, it would seem to be a better solution to provide something like:

```
    @ This code is for reporting errors. If you |#define DEFAULT| when
    compiling this code, the default error display will be used.

    @<Display err...@>=
    #ifdef DEFAULT
    cerr << "!! Error " << errorCode << " encountered.\n";
    #endif
```

That way, if the programmer wants to provide their own WEB additions, they simply have to not define DEFAULT and add a @<Display err...@>= definition, appending their error handling code to the above fragment.

Another solution comes to mind as I'm writing this. The best of all possible worlds, in my opinion, would be to create default error handlers as C macros and provide a mechanism for users to override these macros (in a separate include file, perhaps). This would make for a fully documented standard interface to your code. It would also mean that consumers would never change the code directly. This would look something like this. Early in your file, you would define the user-changeable macros (in the C part or the definition part of your WEB, doesn't matter).

```
    @ These default macros are used throughout the rest of this C++
    class.

    @<Generic Macros@>=
    #define ErrorDisplay(errorCode) \
        cerr << "!! Error " << errorCode << " encountered.\n"

Then you do something like:

    @ User-supplied redefinitions file. If you want to use all the
    defaults, just leave this file empty.

    @<User Redefinitions@>=
    #include "local-hacks.h"    // Redefine ErrorDisplay macro, etc.

And then later, just do:

   @<Display error@>=
   ErrorDisplay(errorCode);

The "local-hacks.h" could contain code like this:

   #undef ErrorDisplay
   #define ErrorDisplay(errorCode) \
   cerr << "Dude!! I seem to have encountered error " << errorCode << "\n"

Hope this helps.
```

From: **Lee Wittenberg**
Date: 28 May 1993

Zdenek Wagner writes: I am writing a C++ pure virtual class in CWEB. The class is intended to be used by other programmers even those who do not know WEB (they will just get the tangled code and printed documentation)and also those who are not very familiar with object oriented programming. Therefore I should supply good information how they will derive their own classes.

And now comes the question. The WEB file should contain examples which should be formatted in the similar way as the C-part. I do not like to study the macros from "cwebmac.tex" and do the formatting by hand. I therefore need some trick to put blocks which will be ignored by tangle and formatted like C by weave. The trick with @(nul@>=... will not work because I intend to leave some modules

unresolved. Imagine the part of the example:

```
 if (errorCode != 0) {
 @<Display error message@>@;
}
```

I will not specify how to display the error message because each programmer has his own way for managing error messages or even his own libraries. It is even more evident on data input and output of results which does not depend upon the properties of the base class but on the nature of the problem. Is there a solution in CWEB?

Even though CWEAVE complains about unresolved modules, it formats them correctly (with a zero for the section number). CTANGLE will also complain, but carry on (expanding nothing into the specified slot, which is what you want). I use this facility quite a bit while building and testing programs in a piecemeal fashion (I did this yesterday, in fact). If the problem is that your make program stops when CTANGLE or CWEAVE exits with a non-zero value, you can do what I do: change the command line in the makefile to begin with a "-". For example:

```
.w.tex:
    -cweave $*
.w.c:
    -ctangle $*
```

As far as I know, the @(nul@>= (or @(/dev/null@>= for Unix types) should work. I've thought about it, but never actually had occasion to try it.

Scraps with explicit arguments

From: **Humberto Zuazaga**
Date: 01 Jun 1993

I was showing some noweb'ed C code to a professor the other day, and he objected to my use of "implicit arguments" in a scrap (or whatever the politically correct term for "module" is these days). He would prefer that scraps could take explicit arguments in their invocation. Do any of the existing litprog tools have this feature? I could make it look like they do by putting the arguments into the scrap name:

@<<process variables (x, y)>>

...

@<<process variables (x, y)>>=

x += y * 23;

but perhaps this is a genuinely usefull feature to have. It looks like it could permit true inline functions in languages that do not support them, and help improve the clarity of webbed code by decreasing reliance on the previously declared variables.

From: **Lee Wittenberg**
Date: 01 Jun 1993

Humberto Zuazaga writes: I was showing some noweb'ed C code to a professor the other day, and he objected to my use of "implicit arguments" in a scrap (or whatever the politically correct term for "module" is these days). He would prefer that scraps could take explicit arguments in their invocation. Do any of the existing litprog tools have this feature? I could make it look like they do by putting the arguments into the scrap name:

@<<process variables (x, y)>>

...

@<<process variables (x, y)>>=

*x += y * 23;*

but perhaps this is a genuinely usefull feature to have. It looks like it could permit true inline functions in languages that do not support them, and help improve the clarity of webbed code by decreasing reliance on the previously declared variables.

As I recall, in one of the articles in his "Literate Programming" book, Knuth recommends mentioning the variables explicitly in the module name. Something like

@<@<Process variables [[x]] and [[y]]>>
...
@<@<Process variables [[x]] and [[y]]>>=
x += y * 23;
in Noweb, or

@<Process variables |x| and |y|@>
...
@<Process variables |x| and |y|@>=
x += y * 23;
in CWEB. This makes the use of the variables explicit, although not automatic. On a related note, I find that global variables often make my webs easier to read than locals. I also find that gotos are much better than breaks and continues (in C code) when the jump occurs in a section separate from the one in which the loop begins (and ends). Of course, this is only true in a webbing system that automatically indexes labels (like CWEB & FWEB), so breaks and continues are probably better in Noweb, unless you explictly use \label and \ref to build your own label index (separate from your ``regular'' index).

From: **David Kastrup**
Date: 02 Jun 1993

As to having arguments to scraps: don't confuse them with procedures! Scraps are usually NOT supposed to be called more than once! They are only used to split a program into several logically units. If you had to include interfaces in them, they would lose their advantage over procedures. In addition, you would be tempted to generate large code pieces, instead of having to parameterize.

The large psychological advantage about WEBs is that you split large tasks pretty automatically without thinking into smaller ones. The moment you think of your code sections of parametrized pieces, you start thinking of interfaces, and that is a work not of much sense considering the small code sections you like to arrive at. If you have parameters, use the mechanisms of the language. That is my feeling.

Are modules necessary?

From: **Stephen McKearney**
Date: 24 Jun 1993

I have developed a WEB for Word system, based around the WORDWEB developed elsewhere, that does almost everything most WEB systems do including : cross-references, code indexes, find definition, etc. Some people here have been using it and we have come to the conclusion that the formal structure of Text Part - Code Part is unnecessary in this WYSIWYG environment. Having code chunks appear throughout the text seems to lead to a slightly different style of programming.

Does anyone else feel that the numbered module structure is more than simply a carry over from the parsing of the source file in the original WEB? Cross-references can be handled using page numbers etc. Before someone goes on about how much better TeX is over Word, etc I should say that if I had a free choice it would be TeX/LaTeX and in my environment most people will not move away from WYSIWYG environments.

From: **Osman Buyukisik**
Date: 24 Jun 1993

Yes, since they allow you develop code without regard to the particular compilers needs (sequence). Does your web has the same limitation as the earlier one namely the 64k limit on the program size (due to the wordbasics string size limit)? Even though I like (La)TeX based webs, I would like to try "word" based ones (If it is available).

From: **Allen Rouse**
Date: 24 Jun 1993

Stephen McKearney said: Does anyone else feel that the numbered module structure is more than simply a carry over from the parsing of the source file in the original WEB? Cross-references can be handled using page numbers etc.

I think I agree with you. When I first heard of Lit. Prog. and read about the subject I developed the impression that the final document would resemble a chapter from a textbook or a book and not a numbered sequence of text/code segments as is currently the case. I imagined that the code would be in its own box (or area) in the document in the same way that photographs and other images are included in newspapers, dictionaries, texts. I also thought that actual images could be included. For example, in a program that draws a line its progress could be graphically shown by pixel diagrams. Or a sorting algorithm could include a graph of its performance on random data. In other words I feel that the ideal product of a Literate Programmer is to be able to produce a final document which is indistinguishable from an ordinary expository/descriptive treatise. Perhaps the best description I could give of the ideal program is, as I first said, "something that looks like a chapter/section from 'Intro. to Algorithms' or another such book"

From: **Eric van Ammers**
Date: 25 Jun 1993

Allen Rouse writes: When I first heard of Lit. Prog. and read about the subject I developed the impression that the final document would resemble a chapter from a textbook or a book and not a numbered sequence of text/code segments as is currently the case. I imagined that the code would be in its own box (or area) in the document in the same way that photographs and other images are included in newspapers, dictionaries, texts. I also thought that actual images could be included. For example, in a program that draws a line its progress could be graphically shown by pixel diagrams. Or a sorting algorithm could include a graph of its performance on random data. In other words I feel that the ideal product of a Literate Programmer is to be able to produce a final document which is indistinguishable from an ordinary expository/descriptive treatise. Perhaps the best description I could give of the ideal program is, as I first said, "something that looks like a chapter/section from 'Intro. to Algorithms' or another such book"

Perfectly true. For exactly this reason I maintain that a literate programming tool should be capable to cooperate with any formatter or wordprocessor one wants to use, without imposing any restrictions on the use of the choosen text processing system. I consider it a severe disadvantage if the literate programmig tool imposes a particular structure of chapters and sections on the documentation or prohibits the use of figures or tables.

From: **Bart Childs**
Date: 27 Jun 1993

I feel strongly that sections are extremely valuable. Can we do without them? Sure! Can we do without the index? Sure! Just look at FunnelWeb, Noweb, and Nuweb. Can we do without good typesetting of the code? Sure! Just look at same list. Can we do without the table of contents? Sure! Then, let us look at the state of software. It is often quoted that the 60 to 80 percent of the cost of a code in its lifetime is maintenance! Code should be done carefully and with full attention paid to the reading aids. I think Don Knuth's design was the result of careful study, consideration of many alternatives, ... that are typical of his works. The reason that Norman Ramsey did Noweb was because of the difficulty of convincing users to try such a complicated system as WEB.

I think the solution is training! WEB like systems offer a reasonable way of weaving documentation and code together to make it easier to maintain... We are embarking on an experiment to use it in beginning courses. We are confident that we will make a difference in their problem solving and programming skills (and we hope writing skills too). Most webs use few of the available web commands and darned few TeX commands too (of course TeX and METAFONT are exceptions). I believe that cooperative efforts to share (like this discussion list) is the way to improve understanding of literate programming. I am pleasantly surprised how many have made good strides by themselves. I had the great pleasure of working with several good graduate students who helped me greatly in understanding the processes.

Have you ever imagined a (say) calculus book without an index, table of contents, ...? It can be such a disaster to try to study and learn a technical subject without such aids. Maintenance programming is often similar to learning a new subject.

From: **Osman Buyukisik**
Date: 27 Jun 1993

I feel strongly that sections are extremely valuable. Can we do without them? Sure! Can we do without the index? Sure! Just look at FunnelWeb, Noweb, and Nuweb. Can we do without good typesetting of the code? Sure! Just look at same list. Can we do without the table of contents? Sure!

I think nuweb has index and table of contents. Only the code typesetting is missing due to its applicability to all languages.

From: **Brett Kotch**
Date: 27 Jun 1993

True that a large part of the time is spent on maintenance, yet that initial time is spent in front of a screen, not looking at a piece of paper. That is where the problem lies. Give me a system which will provide me the benifits of web and allow me to *easily* electronic code and I will buy it.

From: **Norman Ramsey**
Date: 28 Jun 1993

Some people here have been using it and we have come to the conclusion that the formal structure of Text Part - Code Part is unnecessary in this WYSIWYG environment. Having code chunks appear throughout the text seems to lead to a slightly different style of programming.

Does anyone else feel that the numbered module structure is more than simply a carry over from the parsing of the source file in the original WEB? Cross-references can be handled using page numbers etc.

Yes. noweb has no module structure and uses page numbers for cross-reference. I find it infuriating every time I dive into TeX or METAFONT and have to cope with cross-references to module numbers. I confess that noweb does not do page numbers as well as I would like (i.e. 17a, 17b, 17c for chunks appearing on page 17), but I just haven't been able to force myself to try to write even such a simple program using latex's brain-damaged cross-reference mechanism.

From: **Norman Ramsey**
Date: 28 Jun 1993

The reason that Norman Ramsey did Noweb was because of the difficulty of convincing users to try such a complicated system as WEB.

I'm being grossly maligned here (not on purpose, I know, Bart). I created noweb because extensive experience with WEB, CWEB, and Spidery WEB convinced me that WEB should die. I won't go into the details here; they are in my papers. I thought that literate programming was a great idea hiding behind some poor tools. Don Knuth is a brilliant algorithmicist, but his tools can be awkward. For example, the TeX world seems to prefer latex to plain TeX, even though plain TeX is better documented and has fewer bugs. The reason must be that latex comes closer to providing the tools people actually want to use to make documents. The goal of noweb was to eliminate the clutter and awkwardness surrounding the tools so that people could experiment with and evaluate literate programming without having to fight awkward tools.

From: **Norman Ramsey**
Date: 28 Jun 1993

I feel strongly that sections are extremely valuable. Can we do without them? Sure! Can we do without the index? Sure! Just look at FunnelWeb, Noweb, and Nuweb. Can we do without good typesetting of the code? Sure! Just look at same list. Can we do without the table of contents? Sure!

These exclamations are slightly unfair. I can't speak to FunnelWeb (ran screaming from the manual some time ago), but both noweb and nuweb support table of contents and automatic indexing of sections/macros/code chunks (use your favorite term). nuweb has a nice semi-automated solution to the indexing problem. [This reminds me---Preston, I couldn't find where you documented what an identifier is... is it just characters delimited by whitespace?] Even minimalist old noweb supports the latex indexing mechanism.

I think Don Knuth's design was the result of careful study, consideration of many alternatives, ... that are typical of his works.

I'm not so sure I agree. Consider that the part of TeX that is absolutely the most difficult to understand is *not* written as a literate program (I refer to the TRIP test?).

I think the solution is training!

I agree training is needed, but I think we should be teaching people how to use the literate-programming paradigm, not how to use particular tools. Our experience using literate programming on a team project showed us how difficult it was to teach people to write literate programs, and how people can write truly horrendous programs using literate-programming tools. Carla Marceau and I wrestled with this problem over two years, and we thought we had a coherent story about how to write literate programs, but when we started passing our analysis around it fell to bits. We had a good understanding of the issues involved in creating tools, but everything we learned about writing programs can be stated in two words: peer review. I encourage all of you to write down your ideas about how to write literate programs. The world needs an "Introduction to writing literate programs" along the lines of Andrew Birrel's "Introduction to Programming with Threads."

From: **Stephen McKearney**
Date: 28 Jun 1993

I posted a message asking if people thought modules were an important part of the literate programming concept. By modules I mean the format:

1. blah blah blah

<blah> =

code

code

code

The reason for my comment was because some people here have been using a WORD based WEB that I adapted from a previous version. In this WEB we originally had numbered modules but discovered that with numbered sections and chapters the numbering of modules in this strict fashion was not necessary. I am not saying you should remove the code module idea but simply be more flexible over the text.

As I expected the discussion has moved into the 'TeX is best' argument with some people advocating training etc. Having recently moved from a university research environment into an industry research environment I can simply say that, although I like TeX/LaTeX, there is no way the majority of programmers will use TeX when things like WORD are relatively easy to use and look good - fact of life.

As for modules etc. the code we are producing does not seem to need numbering of modules and when you abandon numbering you seem to end up with something closer to a 'book'. I think this is because the numbered module idea seems to set you thinking afresh for each module while without numbers the document feels more like a flowing description. I get this feeling from most literate programming programs I have read that have been published. I am very interested in the views expressed so far.

From: **Jonathan Gilligan**
Date: 28 Jun 1993

Forgive my skepticism, and pardon my impertinence (I've never taught programming above the high-school level, where I used LOGO most illiterately), but to teach literate programming seems as though it would be altogether different from teaching programming and much closer to teaching expository writing. There is no simple how-to for writing an essay that's a joy to read. You can give rules (four-sentence paragraphs, making outlines, the technicalities of footnotes and bibliographies) but the magic is ineffable and the student can only learn it by example (read Orwell's prose, for instance). Similarly, I would, with all humility, suggest that only by publishing excellent literate programs will we give illiterate programmers the tools they need to learn literacy.

Using the various tools themselves seems a small obstacle compared to learning how to think in a literate manner. Thus, I find completely plausible Norman Ramsey's statement that nothing helped except peer review. Nothing else helps novelists or poets either.

From: **Preston Briggs**
Date: 28 Jun 1993

Stephen McKearney writes: I posted a message asking if people thought modules were an important part of the literate programming concept. By modules I mean the format:

1. blah blah blah

<blah> =

code

code

code

Sure, I (and others) agree with your points. I write text, organized however I feel is best, with chapters, sections, figures, tables, and everything else LaTeX provides. I also happen to have code organized into scraps which are cross-referenced, indexed, etc. This is all in the context of nuweb. Noweb and FunnelWeb have similar feature sets. Nuweb does all the cross references in terms of scrap numbers. Noweb does it all in terms of pages numbers. I'd prefer page numbers; but I couldn't figure out how to do it all as cleanly as I wanted. Presumably others will reflect on their experience (and ours) and will design new tools with different features that better support their concept of literate programming. Maybe we'll reach a consensus or maybe not; I doubt it matters.

From: **Zdenek Wagner**
Date: 28 Jun 1993

Bart Child writes: I think the solution is training! WEB like systems offer a reasonable way of weaving documentation and code together to make it easier to maintain... We are embarking on an experiment to use it in beginning courses. We are confident that we will make a difference in their problem solving and programming skills (and we hope writing skills too). Most webs use few of the available web commands and darned few TeX commands too (of course TeX and METAFONT are exceptions). I believe that cooperative efforts to share (like this discussion list) is the way to improve understanding of literate programming. I am pleasantly surprised how many have made good strides by themselves. I had the great pleasure of working with several good graduate students who helped me greatly in understanding the processes.

Take for example the book by Niklaus Wirth: Algorithms + Data Structures = Programs. This book shows how the program should be designed. It is written in conventional Pascal but the way how the program grows in the examples is exactly the same as in WEB. Using WEB in the courses for beginners (even those who do not know TeX) will be easy provided TeX is already installed.

From: **Zdenek Wagner**
Date: 28 Jun 1993

Allen Rouse writes: When I first heard of Lit. Prog. and read about the subject I developed the impression that the final document would resemble a chapter from a textbook or a book and not a numbered sequence of text/code segments as is currently the case. I imagined that the code would be in its own box (or area) in the document in the same way that photographs and other images are included in newspapers, dictionaries, texts. I also thought that actual images could be included. For example, in a program that draws a line its progress could be graphically shown by pixel diagrams. Or a sorting algorithm could include a graph of its performance on random data. In other words I feel that the ideal product of a Literate Programmer is to be able to produce a final document which is indistinguishable from an ordinary expository/descriptive treatise. Perhaps the best description I could give of the ideal program is, as I first said, "something that looks like a chapter/section from 'Intro. to Algorithms' or another such book"

Eric W. van Ammers writes: Perfectly true. For exactly this reason I maintain that a literate programming tool should be capable to cooperate with any formatter or wordprocessor one wants to use, without imposing any restrictions on the use of the choosen text processing system. I consider it a severe disadvantage if the literate programming-tool imposes a particular structure of chapters and sections on the documentation or prohibits the use of figures or tables.

Who says that WEB imposes a particular structure of chapters? As I know CWEB 2.7 (I have not retrieved CWEB 3 yet), it suggests a structure which is probably used by everybody. You can change it if you redefine macros in cwebmac.tex. CWEB 3 is probably most flexible. My documentation includes tables and figures because it is supported by TeX. If you know TeX, you know how to do it.

From: **Osman Buyukisik**
Date: 29 Jun 1993

Hi, maybe I misunderstood the first time, but it looks like you are right "numbers" are not needed. Need a better method than just page numbers for the indices (like what Norman said 17a, 17b,...). Is that possible in WORD? Are you going to have it availble? PS: you have not answered about the code size restriction (<64k) of the earlier wordweb being still true?

From: **Stephen McKearney**
Date: 29 Jun 1993

Osman Buyukisik writes: Hi, maybe I misunderstood the first time, but it looks like you are right "numbers" are not needed. Need a better method than just page numbers for the indices (like what Norman said 17a, 17b,...). Is that possible in WORD? Are you going to have it availble? PS: you have not answered about the code size restriction (<64k) of the earlier wordweb being still true?

I never thought of indexing like this although I'm not sure it will be much more useful than just page numbers.

The code size restriction is just on generated code, that is, the output of the tangle phase. I do not find this to be a problem because I tend to keep individual WEB files reasonably small and WORD provides facilities to create large documents from a number of files. For example, if you write a large thesis you might have files called CHAPTER1.DOC, CHAPTER2.DOC, etc. A more serious problem was caused because the main tangle routine in the original version used recursion that VERY quickly ran out of memory. Judicious use of GOTOs etc. removed the recursion and it seems to handle reasonably large files now. If people are interested in this version I can look into supplying it but I do not have FTP access.

From: **Afzal**
Date: 29 Jun 1993

Allen Rouse writes: In other words I feel that the ideal product of a Literate Programmer is to be able to produce a final document

which is indistinguishable from an ordinary expository/descriptive treatise. Perhaps the best description I could give of the ideal program is, as I first said, "something that looks like a chapter/section from 'Intro. to Algorithms' or another such book"

I still expect something similar, as I haven't yet started using literate programming. I have been reading through this discussion list and from literature about Literate Programming. I am a (Fortran based) software developer, prefer to concentrate more on the design phase; with 2B Lead pencil, rubber and bunch of papers; as only tools. This phase does involve the text, figures and blocks of code. It takes me very small time in implementation, but a good system for literate programming could reduce it even further, and save me from rewriting/puting it into computer.

Although the man-made systems and products are always evolving in nature and need improvement, but still if some product could prvide some mechanism to embed code, text and graphics; and with minimum learning overhead would be much appreciated. One such system has been proposed in recent issue of Software Engineering Journal: Stephen Shum and Curtis Cook AOPS: an Abstraction-Oriented Programming System for Literate Programming Software Engineering Journal, V 8, n 3, pp 113-120, May 1993 They claim, AOPS is text processor and programming language independent, and can embed code, text and graphics in same source AOPS file. It does have its browser and other tools. It is PC-based. I have requested its copy from authors, which they have offered free to anybody who (sends them a diskette and) wants to use it.

From: **Norman Ramsey**
Date: 01 Jul 1993

Putting implementation issues aside -- what is the most convenient label of modules for using a Web?

In my experience reading TeX and METAFONT, module numbers drove me nuts. I propose labelling each code chunk with the page number of its first definition. If more than one code chunk is defined on a page, the page numbers should have lower-case letters appended; e.g., 17a, 17b, 17c.

From: **Lee Wittenberg**
Date: 03 Jul 1993

Stephen McKearney writes: I have developed a WEB for Word system, based around the WORDWEB developed elsewhere, that does almost everything most WEB systems do including : cross-references, code indexes, find definition, etc.

As the author of the original (primitive) WORDWEB system, I would be interested in seeing what you came up with.

Some people here have been using it and we have come to the conclusion that the formal structure of Text Part - Code Part is unnecessary in this WYSIWYG environment. Having code chunks appear throughout the text seems to lead to a slightly different style of programming.

Does anyone else feel that the numbered module structure is more than simply a carry over from the parsing of the source file in the original WEB? Cross-references can be handled using page numbers etc.

I agree. My noweb style is quite different from my CWEB style. I rely on LaTeX for structuring noweb programs. Each style has something to recommend it, though. It's a lot easier to find a referenced chunk when the reference refers to a section rather than a page (noweb's noxref generates page references; I've experimented with using \ref instead of \pageref, but unless each section is kept very small, it's harder to deal with than the page numbers).

Before someone goes on about how much better TeX is over Word, etc I should say that if I had a free choice it would be TeX/LaTeX and in my environment most people will not move away from WYSIWYG environments.

You might be interested in my experiences at Tipton Cole + Co. using WORDWEB. A few people (who were diehard WYSIWYGers) started using WORDWEB (including Tipton, the boss), but found that the advantages of noweb/LaTeX far outweighed the advantages of the WYSIWYG world. TCC seems to be moving in the direction of a "literate programming shop," using noweb. I'd be glad to put my boss in contact with your boss, if you think it would help.

From: **Stephen Fulling**
Date: 06 Jul 1993

Stephen McKearney writes: As for modules etc. the code we are producing does not seem to need numbering of modules and when you abandon numbering you seem to end up with something closer to a 'book'. I think this is because the numbered module idea seems to set you thinking afresh for each module while without numbers the document feels more like a flowing description. I get this

feeling from most literate programming programs I have read that have been published.

It occurred to me that this difference in style and taste is related to the difference in the way mathematicians, as opposed to mathematical scientists, write papers and books. Hard-core mathematics is written in little chunks, often numbered, and always labeled "Definition", "Theorem", "Proof", "Example", etc. Theoretical physicists write ordinary, discursive, stream-of-consciousness prose frequently interrupted by displayed equations.

This difference has something to do with subject matter [How often does a physicist genuinely prove a theorem? :-)] but it also has a lot to do with habit, tradition, and taste. Being raised as a physicist, I have to make a conscious effort to remind myself to write in the mathematical style; but I certainly recognize its advantages in certain circumstances.

To my mind, one of the advantages of the modularized LitProg paradigm is precisely that it "sets you thinking afresh" by isolating each main idea or task in a separate piece, while also showing how that piece plugs into the larger structure. But, just as in my physics-oriented papers, there may be circumstances where the "flowing description" is more appropriate.

Transportable webs in SGML

From: **Edward Keith**
Date: 1 Jul 1993

If the purpose of LitProg is to communicate what we have done, then this is a clear failure! Marcus has a web that is undecipherable to me, and even if I manage to figure out this problem, that simply postpones the pain until next time. So, the question is, in general, how do we make webs (and other tex files) transportable?

SGML? It is an ISO standard.

From: **Chris Flatters**
Date: 1 Jul 1993

SGML is changing the playing field a little from TeX. SGML is a markup language whereas TeX is a typesetting language. SGML describes the structure of a document (eg. the following word is an identifier) rather than its appearance (eg. the following word should appear in italic type). It would certainly make sense to have a WEB system that wove to SGML since we should really be more concerned with the content of a woven document than its appearance. It might be even more useful if it generated HTML (a version of SGML with hypertext extensions used for the World-Wide Web project).

From: **Karl Vogel**
Date: 2 Jul 1993

Chris> It would certainly make sense to have a WEB system that wove to SGML Chris> since we should really be more concerned with the content of a woven Chris> document than its appearance.

This definitely gets my vote. The nicest thing about SGML is the fact that parsers for it are freely available. I can write something to turn valid SGML into Texinfo or Troff a lot more easily than I can write something to manipulate (say) a given style of WEB code. The best thing about SGML is that it can be used to generate several representations of any document. You can have a driver that reads an SGML document and generates output suitable for printing, or ready-to-compile code, or a hypertext representation of your program intended for easy browsing.

It might be even more useful if it generated HTML (a version of SGML with hypertext extensions used for the World-Wide Web project).

I have one minor quibble with this idea. How stable is the HTML document definition? It won't help us if the rules for creating an HTML document are too fluid. We might be better off creating a document definition which is solely intended for literate programming types. This way, we have more control over our own destiny; the HTML folks can do what they like, and it won't hurt us a bit. I'm not saying that we couldn't borrow ideas from them; I just don't think we should tie our destiny to a style of document representation that's rooted in information retrieval rather than programming.

From: **Trevor Jenkins**
Date: 2 Jul 1993

Back in the days before the ISO 8879 (SGML) was published I was a member of the British committee that participated in the work. For the technical report on using SGML I had proposed that an example SGML DTD be written which would allow WEB files to be analysed. I even volunteered to write it. My fellow committee members didn't see WEB as sophisticated enough---they wanted a DTD for Z instead. Not being a Z expert I dropped the idea. Their interest in Z was such that they didn't do anything either. On reflection I wished that I had persevered with the WEB one. :-(

From: **Jeffrey McArthur**
Date: 2 Jul 1993

SGML is changing the playing field a little from TeX. SGML is a markup language whereas TeX is a typesetting language. SGML describes the structure of a document (eg. the following word is an identifier) rather than its appearance (eg. the following word should appear in italic type).

Just a few comments about SGML. First, there are a lot of misconceptions about SGML. I know, I have had to learn a tremendous amount in the past few weeks since I am now working on a massive SGML job. TeX and SGML go together very nicely. I am currently feeding raw SGML files into TeX and typesetting them. If are willing to write some macros and play with catcodes you can relatively easilly typeset SGML documents with TeX without the need for any pre-processor. The much more serious problem with SGML is that it is almost impossible to edit. Let me give you an idea. This is from a real sgml document that am printing via TeX:

```
<INGRDNTS ID="S2" NUMBER="2"><TITLE>HAZARDOUS INGREDIENTS
</TITLE><SUBSECT><TITLE>HAZARDOUS INGREDIENTS</TITLE><TABLE>
<TBLHEAD><TBLBODY><TBLCOLS><TBLCOL HALIGN="Justify"><TBLCOL
 HALIGN="Justify"><TBLCOL HALIGN="Justify"><TBLCOL
 HALIGN="Justify"></TBLCOLS><TBLROW><TBLCELL COLSTART="1"
 HALIGN="Center">Hazardous Components</TBLCELL><TBLCELL
 COLSTART="2" HALIGN="Center">OSHA PEL</TBLCELL><TBLCELL
 COLSTART="3" HALIGN="Center">ACGIH TLV</TBLCELL><TBLCELL
 COLSTART="4" HALIGN="Center">CAS Number</TBLCELL></TBLROW>
</TBLBODY></TBLHEAD><TBLBODY><TBLCOLS><TBLCOL HALIGN="Justify">
<TBLCOL HALIGN="Justify"><TBLCOL HALIGN="Justify"><TBLCOL
 HALIGN="Justify"></TBLCOLS><TBLROW><TBLCELL
 COLSTART="1">Calcium Carbonate</TBLCELL><TBLCELL
 COLSTART="2">3.75 mg/m<SUP>3</SUP> 1.4 mg/m<SUP>3</SUP> resp.
 dust</TBLCELL><TBLCELL
 COLSTART="3">10 mg/m<SUP>3</SUP></TBLCELL><TBLCELL COLSTART="4">1317-65-3
</TBLCELL></TBLROW><TBLROW><TBLCELL COLSTART="1">Pyrophyllite
</TBLCELL><TBLCELL COLSTART="2">50 mppcf</TBLCELL><TBLCELL
 COLSTART="3"></TBLCELL><TBLCELL COLSTART="4">12269-78-2
</TBLCELL></TBLROW><TBLROW><TBLCELL COLSTART="1">Muscovite
 (MICA)</TBLCELL><TBLCELL COLSTART="2">20 mppcf</TBLCELL>
<TBLCELL COLSTART="3"></TBLCELL><TBLCELL COLSTART="4">1318-94-1
</TBLCELL></TBLROW><TBLROW><TBLCELL COLSTART="1">Kaolinite
</TBLCELL><TBLCELL COLSTART="2">50 mppcf</TBLCELL><TBLCELL
 COLSTART="3">0.1 mg/m<SUP>3</SUP></TBLCELL><TBLCELL COLSTART="4">1332-58-7
</TBLCELL></TBLROW><TBLROW><TBLCELL COLSTART="1">Quartz (total)
</TBLCELL><TBLCELL COLSTART="2">30 mg/m<SUP>3</SUP> /(&percnt; quartz +2)
</TBLCELL><TBLCELL COLSTART="3">0.1 mg/m<SUP>3</SUP></TBLCELL><TBLCELL
 COLSTART="4">14808-60-7</TBLCELL></TBLROW></TBLBODY></TABLE>
</SUBSECT></INGRDNTS>
```

This is an extreem example. Part of the problem is caused because SGML does not normally break the file into separate lines. It is not unusual to run into a 200K SGML file without a single line feed (or carridge return, or cr/lf, or carridge control, all depending on what OS you are running under). This breaks a lot of tools (including TeX). Another thing to realize is that it is possible to have commands for italic, bold, and so on in an SGML file. <ITALIC> is not an uncommon tag. Also the move verbose <EMPH TYPE="ITALIC"> is also found. Now the first example was a bit nasty because it had a table in it. So here is a second example, from the same document, that is a bit easier to deal with:

```
<OTHER ID="S8" NUMBER="8"><TITLE>GENERAL CONTROL MEASURES
</TITLE><SUBSECT ID="S8-1"><TITLE>Ventilation</TITLE><PARA>None
 other than normal with ordinary use.</PARA></SUBSECT><SUBSECT
 ID="S8-2"><TITLE>Respiratory Protection</TITLE><PARA>None with
 ordinary use. If prolonged exposure, wear a MSHA/NIOSH
```

```
 approved dust/pesticide respirator. Avoid breathing dust.
</PARA></SUBSECT><SUBSECT ID="S8-3"><TITLE>Protective Gloves
</TITLE><PARA>None with ordinary use. If handling spill, wear
 impervious gloves.</PARA></SUBSECT><SUBSECT ID="S8-4">
<TITLE>Eye Protection</TITLE><PARA>None with normal use. Wear
 safety glasses with side shields or goggles if eye contact is
 possible.</PARA></SUBSECT><SUBSECT ID="S8-5"><TITLE>Other
 Protective Clothing and Equipment</TITLE><PARA>To avoid contact
 with skin, wear protective apparel during application.</PARA>
</SUBSECT></OTHER>
```

This is much more code intensive than TeX. This is why SGML tools try and and allow you to edit without the tags (Author/Editor for example). One final thought. SGML can be viewed as a meta-markup-language. You can define you own mark-up scheme in SGML if you are willing to change the characterset (allowed in SGML) and so on. I bet, if you were seriously ambitious, you could almost write a DTD for WEB (classic Pascal WEB).

From: **Trevor Jenkins**
Date: 3 Jul 1993

Just a few comments about SGML. First, there are a lot of misconceptions about SGML. I know, I have had to learn a tremendous amount in the past few weeks since I am now working on a massive SGML job.

SGML has always been plauged by this problem. During my stint in the international standards work the problem was from the office people (ODA/ODIF ISO 8613) who felt threatened tht SGML was encroaching upon their remit. It doesn't in that SGML is aimed at high-end publishing where as ODA/ODIF was aimed at the typical character-cell based office environment (_personally_ I always felt that what ever the ODA crew were adding into their architecture SGML already did and better. :-)

The much more serious problem with SGML is that it is almost impossible to edit. Let me give you an idea.

That isn't really a problem with the Standard Generalized Markup _Language_. If it were then one could make the same complaint about programming languages, eg C, Pascal or even (dare I say it) Literate Code. The work that Mike Cowlishaw (IBM) did for the OED project demonstrated that an SGML oriented editing system ca be created and use with a very complex document.

....Part of the problem is caused because SGML does not normally break the file into separate lines.

Again it does need to. Consider the output from TANGLE where there is the same problem. If you automated text-generation then you will end up with output that is impossible for human beings to read.

It is not unusual to run into a 200K SGML file without a single line feed (or carridge return, or cr/lf, or carridge control, all depending on what OS you are running under). This breaks a lot of tools (including TeX).

I have received mail and news message which break my mailer/news-reader that is not the fault of the message (which by the way didn't exceed the minimum limits imposed by the appropriate RFCs) but rather laziness on the part of the programmer. (I didn't really mean to insult DEK by that comment :-)

Another thing to realize is that it is possible to have commands for italic, bold, and so on in an SGML file. <ITALIC> is not an uncommon tag.

Then it is wrong.

Also the move verbose <EMPH TYPE="ITALIC"> is also found.

This is the purist-style and is indeed what the text of ISO 8879 mandates. Not that it is possible to check for it.

One final thought. SGML can be viewed as a meta-markup-language. You can define you own mark-up scheme in SGML if you are willing to change the characterset (allowed in SGML) and so on.

SGML includes both a mechanism for specifying the structure of a document (your meta-markup) AND how a document should be encoded so as to conform to that structure. With public entity texts it is possible to create an SGML that doesn't include the specific DTD being used though it must be available when the document is processed.

I bet, if you were seriously ambitious, you could almost write a DTD for WEB (classic Pascal WEB).

I don't think that the task is that "ambitious". Nor do I think that it need to be limited to classic Pascal WEB. Now I have to go off and prove it dont I. ;-)

From: **Joachim Schrod**
Date: 5 Jul 1993

[This is a long response, sorry. I tried to cut it down, but I cannot make it less text without loosing clarity.]

SGML is changing the playing field a little from TeX. SGML is a markup language whereas TeX is a typesetting language. SGML describes the structure of a document (eg. the following word is an identifier) rather than its appearance (eg. the following word should appear in italic type).

Jeffrey wrote: Just a few comments about SGML. First, there are a lot of misconceptions about SGML. I know, I have had to learn a tremendous amount in the past few weeks since I am now working on a massive SGML job.

Please, don't take this mail personally -- but it's my impression that you have more to do. Your mail gives IMNSHO a completely wrong presentation of SGML. In addition,

TeX and SGML go together very nicely. I am currently feeding raw SGML files into TeX and typesetting them.

Yes, that can be done for particular document types -- but it's really not the "canonical" way. TeX isn't a programming language that's really suited for this type of tasks. Let me get a bit more structured, so that you can critizice me, too ;-): First I'll give an outline what SGML is. Then I'll attack your statement that SGML documents are unreadable, both in principle and with an example. At last I'll do a sketch of the `usual' connection with SGML and TeX.

WHAT IS SGML?

SGML is an acronym, it stands for Standard Generalized Markup Language. The important word herein to distinguish it from other markup languages is ``Generalized''. We can distinguish four categories of markup:

1. presentational markup - The document's structure is shown by laying out the content on the page/screen. This might seem trivial (introducing spaces and lines) and might be more (itemized lists, etc.) This is what I'm doing manually now...

2. procedural markup - The text is interspersed with formatting commands, which explain how the document is to be formatted. (plain) TeX and [nt]roff are typical examples of this.

3. generic or descriptive markup - The document is tagged to show its structure explicitely. The tags are defined externally. Scribe is the canonical example for this category. From its intent, LaTeX belongs here, too. But one has still the full access to the procedural facilities of TeX. (IMO the greatest strength and the greatest weakness of LaTeX.)

4. generalized markup - The document is described on three levels, which build the parts of a complete document. The first level tells how the input will look like. It will describe the character set, what interpretations are associated with characters, how tags are created, etc. I.e., one explains the lexical conventions, how lexems to describe the document are built. The second level defines the potential structure of the document. One explains explicitely which structural elements may occur and which relationships and attributes they have to another (consists-of, is-optional, etc.) I.e., one defines a grammar which explains the document structure. The third level is the document's content, tagged according to the conventions introduced on level 1 and 2. This part is called document instance in SGML terminology.

SGML belongs to category 4. In so far as it contains a language to **define** markup languages, it is more than "yet another markup language." That's the reason why it's called a meta-language by some people. That's arguable, 'though I won't subscribe to this view -- we describe full documents after all, and full documents have no `meta'-ness whatsoever.

READABILITY OF DOCUMENT INSTANCES

Jeffrey presented horrible examples of SGML input. The problem here is that one has to distinguish between two ways of working with SGML documents: the "wealthy way" (with appropriate tools) and the "poor man's way" (by hand). If the example concerned the former way it was simply wrong; if it concerned the latter, it was plain bad. The wealthy way uses context-sensitive editors, embedded in author systems. The author will (should ;-) never see the tagged text. As somebody already noted, it's like not looking at the output of TANGLE. So the readability of the internal [sic!] document representation -- as presented by Jeffrey -- is not of any concern here. The readability of the document as presented by the editor is the point to ask for. And here you don't see that mess, you'll see a nice

presentation of your structure, with outlining possibilities and all kind of things you dream of (querying for the contents of specific elements and similar things). The poor man's way uses a simple editor and types in the markup declaration and the document instance. Then he or she will add shortcuts which makes typing more easily. Tags can be ommitted from the document instance, the SGML system will insert them for your convenience. As a real life example, a document instance taken (almost verbatim, minus the DTD and some ommitted text declaration from a file here:

```
----------------
<itiman>

<headline>TUD/ITI
<name>sman
<chapter>1
<whatis>convert SGML itiman manual page to nroff man format

<synopsis>
<synline>sman [-esis] <em>file[.sgm]</>
</synopsis>

<description>

<mref>sman</> provides an easy way for converting manual pages in
SGML itiman format to nroff (conversion to TeXinfo is planned).
Either SUN and IBM/HP man package format is created automatically.

Two different kinds of document structures are supported: one for
command man pages that consist of sections like synopsis,
description, options, etc., and one for miscellaneous ones which only
consist of sections with arbitrary names.

For both, the text can contain markups for emphasized text,
description and option lists, verbatim mode, and more. A detailed
description of the markup can be found in the tutorial ``<em>How to
write a SGML itiman Manpage</em>''.

</description>

<options>
<optionlist>
   <option>     -esis
   <optiondesc> generate only the intermediate representation as created by
                the sgmls frontend
</optionlist>
</options>

[...]

<seeAlso>
<mref>sgmls(1)</>, <mref>nroff(1)</>
</seeAlso>

</itiman>
----------------
```

To cite Jeffrey: The much more serious problem with SGML is that it is almost impossible to edit. Let me give you an idea.

I find the example above neither unreadable nor impossible to edit. My idea is obviously different from yours -- the readers of this mail should judge for themselves. (I.e., the created nroff source is IMHO much more unreadable... [this is only a partly extraction])

```
----------------
.st "sman" 1 "TUD/ITI" \*(]W
.SH NAME
sman \- convert SGML manual page to nroff man format
.SH SYNOPSIS
.B sman [-esis] \fIfile[.sgm]\fR
.br
```

```
.SH DESCRIPTION
\fBsman\fR provides an easy way for converting manual pages in
SGML itiman format to nroff (conversion to TeXinfo is planned).
Either SUN and IBM/HP man package format is created automatically.
.PP
Two different kinds of document structures are supported: one for
command man pages that consist of sections like synopsis,
description, options, etc., and one for miscellaneous ones which only
consist of sections with arbitrary names.
.PP
For both, the text can contain markups for emphasized text,
description and option lists, verbatim mode, and more. A detailed
description of the markup can be found in the tutorial ``\fIHow to
write a SGML itiman Manpage\fR''.
.SH OPTIONS
.TP
.B      -esis
generate only the intermediate representation as created by
the sgmls frontend
----------------
```

PROCESSING SGML DOCUMENTS OR _THE CONNECTION TO TeX_

If are willing to write some macros and play with catcodes you can relatively easilly typeset SGML documents with TeX without the need for any pre-processor.

The pre-processor -- usually called SGML parser -- is exactly the tool which makes SGML so valuable. It delivers a canonical form of the document instance, where all ommitted tags and all shortcuts are expanded. It checks the validity of the markup, i.e., one is sure afterwards that the document is correctly tagged. Therefore it is easy to transform this now into valid TeX markup. To program this validation and this `normalizing' in TeX itself is a nightmare. (So much about the statement of Dominique that companies are more concerned about maintainance, modularity, and the ability to support it by other people, than universities. :-) That it is doable, doesn't mean this work is well invested. (IMO, of course. But I'm programming in TeX for 11 years now, and know its limitations quite well.)

If you have enough money and work under UNIX systems -- have a look at the SGML Publisher of Arbortext. (I have no connection to this company except knowing a few people there personally.) It uses an extended version of TeX as the publishing engine to SGML documents. It's a great piece of software for professional document preparation. Especially the table and math support is great (where traditionally SGML systems were weak). Author/Editor -- mentioned from Jeffrey already -- is a nice system. DynaText from Electronic Book Technologies is often mentioned as outstanding, 'though I hadn't the chance to look at it yet. On the freely distributable (poor man's ;-) side: There are some converters available, most notably gf and Format. A large archive of freely distributable material is accessible by anonymous ftp at "ftp.th-darmstadt.de [130.83.55.75] directory pub/text/sgml" In the subdir documentation/ there's also the c.t.sgml FAQ and the famous SGML bibliography of Robin Cover.

From: **Jeffrey McArthur**
Date: 7 Jul 1993

Let me respond to a few points Joachim raised.

... TeX isn't a programming language that's really suited for this type of tasks.

Actually I find TeX to be extreemly well suited for this task. It is much better than any alternative I can find. Let me give you a bit of history about the data I presented. The data was given to us in this fashion. We received over 5000 separate SGML files. Each of them coded this way. Only 10% of the files parsed. This job has been a nightmare. I have had to work for several weeks just trying to get the data to parse and resemble something reasonable. (To give you an idea, as a parting shot the previous contractor replaced all the occurances of "oc" in all the files with the hex character A1 followed by an uppercase C.)

Your arguments are fine if you are working with rational people who present you rational data. We are not. We received data that has MANY problems. Including entire sections miss-coded and so on. This is inspite of the problem that the original DTD was very, very sloppy. We have done a lot of work re-writing the DTD so that it has some semblance of order. Let me explain it this way. If you are writing from scratch and using good tools, SGML is a wonderful tool. If you are handed 50 Meg of data which the client claims to be SGML (but only 10% of it parses) and you have to deal with it in whatever way you can, well it is not a pleasant task.

Jeffrey presented horrible examples of SGML input. The problem here is that one has to distinguish between two ways of working with SGML documents: the "wealthy way" (with appropriate tools) and the "poor man's way" (by hand). If the example concerned the former way it was simply wrong; if it concerned the latter, it was plain bad.

What would you do if you were give 5000 files, of which only 500 parsed, and you had to edit the data? For example, Author Editor will NOT read in many of the files because they are coded wrong. You have no choice but to use the ``poor man's way''. You have to look at the coding. You also need a parser which will show you the problems.

Context-sensitive editors work great if the data parses. Consider a common problem we run into. We need to create an SGML document from a hard copy source. There is no electronic copy of the data. We have two options: scanning and keying. Scanning is fine, but you still have to add all the tags. Keying can have both done at the same time. Most keying houses can easilly add the SGML tags to a document. However, in both cases you will end up with documents that probably will not parse on the first pass. Also with keying you can run into some structure problems, that is, the tags are wrong in such a way that it conficts with the DTD. In that case, there is no option, but the "poor man's way". One of the serious problems with all the editing tools for SGML is that they assume that the data is tagged in a reasonable in accordance with the DTD. This is what you want to end up with, but you may not start with anything resembling that.

One of the serious problems with SGML is that the name has almost become a "buzzword". Not quite as bad as "object-oriented" but it may get there. We have another client who is moving in the direction of SGML. The data is not there yet. They don't want to spend the time nor the money to try and validate the 100 or so Meg of data that they have. But the tags are now consistant with SGML style, and it may be possible to write a DTD for what they have. But we know that the data would not parse. There are too many inconsistancies in the data. It would take a lot of time and effort to get all the data to pass a parser. But eventually they will do that, but not this year, nor next year.

So what do you do if you receive a Pseudo-SGML document? In our case we MUST deal with it. We try and point out the problems with the document to our clients. We will try and fix them, if we can. So if you live in a world where you only deal with real SGML documents that always parse, you have it easy. I do not. I must deal with documents that don't parse, that are miss-tagged, and have structural errors in them. When I am through with them, they parse.

WEB in a business context

From: **Dominique Dumont**
Date: 02 Jul 1993

John Krommes writes: Regarding this latter issue, I feel that if a language such as Fortran or C is supported, use of a language-sensitive processor is far superior to an insensitive one, for the reasons mentioned recently by Bart Childs. However, it is clearly impractical to support all possible languages, so tools such as NUWEB certainly have their place. I expect that the next release of FWEB will offer a "no-language" option, implemented as one of FWEB's multiple languages. This is a straightforward extension of FWEB's meta-comment facilities and will enable one to move between language sensitivity and NUWEB-type output at will.

Since I work in a company, I have other mandatory requirements to be able to use a Literate programming system : (1) The new method must be totally transparent to others people in my group, so I must be able to tangle the web into a readable code with a certain amount of comments (I don't know how much or if I can let TeX commands in the comments of the new generated source). Maybe I should also be able to mix classical program and literate ones. (2) The code must be supportable by other people (others engineers, technical marketing people) without a web system. (3) I must be able to use it along with others software development tools : softbench, debuggers (I don't know yet what are the impacts for the web tools)

One future feature is to be able to generate man pages from the web along with the code and the tangled doc. Without that I will only be able to use noweb if I want to program in a literary way. From what I read on this list, people from university have different requirements for their coding style compared to factory people. University guys tend to write monolithic programs which are seldom reused and are used by a few people. (University guys can flame at will if I'm wrong :-))

Whereas we write programs composed of several chunks written by different people with different coding styles (sometimes with differents languages). Furthermore our programs are sold so they are used by a huge amount of people for several years sometimes tens of years. So the program must be designed to be still upgradable in 10 or 20 years. Nobody knows what will be left of WEB systems in the next century, and we can't afford to support public domains tools , so we cannot take the risk to write programs which depends on WEB to be upgraded. So at the end of a developemnt I must provide a readable source codes with consistents comments. (I don't think some LaTeX command here and there, or line numbers comments are a problem, The aim is that the program must be understandable without WEB tools).

In fact, what my boss says is that I can use WEB if it transparent to other people. What do you, factory people, think of a WEB system in a business context? What are your requirements to be able to use it?

From: **Lee Wittenberg**

Date: 03 Jul 1993

Dominique Dumont writes: Since I work in a company, I have other mandatory requirements to be able to use a Literate programming system : (1) The new method must be totally transparent to others people in my group, so I must be able to tangle the web into a readable code with a certain amount of comments (I don't know how much or if I can let TeX commands in the comments of the new generated source). Maybe I should also be able to mix classical program and literate ones. (2) The code must be supportable by other people (others engineers, technical marketing people) without a web system. (3) I must be able to use it along with others software development tools : softbench, debuggers (I don't know yet what are the impacts for the web tools)

I've just spent the best part of a year trying to introduce literate programming techniques (via noweb) in a business environment. noweb has the "nountangle" tool to help meet your first criterion. The second two have turned out not to be quite so important, as we have been programming for Paradox, a system notoriously short of decent tools. However, we have been able to use the "standard" Unix tools (awk, grep, make, etc., in DOS versions) to ease software development somewhat.

One future feature is to be able to generate man pages from the web along with the code and the tangled doc. Without that I will only be able to use noweb if I want to program in a literary way.

We use a slightly modified manpage.sty file. All our webs follow pretty much the same structure (LaTeXish commands):

```
\maketitle  % a title page followed by a copyright/version control info page
\makecontents  % a table of contents
\part{Interface}
     % manual pages go here
\part{Implementation}
     % the web proper goes here
```

From what I read on this list, people from university have different requirements for their coding style compared to factory people. University guys tend to write monolithic programs which are seldom reused and are used by a few people. (University guys can flame at will if I'm wrong :-))

I'm a university guy, but I understand what you're saying here.

Whereas we write programs composed of several chunks written by different people with different coding styles (sometimes with differents languages). Furthermore our programs are sold so they are used by a huge amount of people for several years sometimes tens of years. So the program must be designed to be still upgradable in 10 or 20 years. Nobody knows what will be left of WEB systems in the next century, and we can't afford to support public domains tools , so we cannot take the risk to write programs which depends on WEB to be upgraded. So at the end of a developemnt I must provide a readable source codes with consistents comments. (I don't think some LaTeX command here and there, or line numbers comments are a problem, The aim is that the program must be understandable without WEB tools).

The people I've been working with are just starting on the literate programming experiment, but seem committed to it. noweb has two advantanges here: (1) it's made up of several very simple tools that are quite easy to maintain yourself (I speak from experience here -- I had to port noweb to MS-DOS, which was surprisingly simple from the noweb side, although a bit of a pain due to DOS restrictions -- I've found it quite easy to to hack a new feature to noweb the few times I've needed to), and (2) you can use nountangle to "undo" your webs into (more or less) normally documented programs if the experiment fails.

In fact, what my boss says is that I can use WEB if it transparent to other people.

What I did was just to use noweb for my stuff and pass the programs around for comment. I finally got one of my coworkers to try it as well (a skeleton web and a one page summary of the necessary commands -- @ and <<>> for noweb; \section (etc.), \em, and the special TeX characters for LaTeX -- were all that was needed to get him started). He was soon converted, and that got everyone else interested. After a brief fling with WinWordWEB, the shop seems to have settled on noweb. It seems to have been a classic example of what Grace Hopper used to refer to as "the necessity of educating our bosses" (and coworkers, in this case). literate programming seems to be one of those things that seem to be more trouble than they're worth until you try them -- then you wonder how you got along without them. Anybody else remember the effort it took to abandon your trusty line-oriented editors and use a screen editor?

What do you, factory people, think of a WEB system in a business context? What are your requirements to be able to use it?

I, too, would like to hear from "factory people."

Tools/Techniques needed before wide acceptance of literate programming?

From: **Guy Bailey**
Date: 14 Jul 1993

I would be in favor of a newsgroup, but I don't think the tools or techniques (especially not the techniques) have reached that stage yet, so I prefer to stay in a ghetto if possible.

What tools or techniques do you think we still need before literate programming is 'ready for the masses'?

From: **Norman Ramsey**
Date: 14 Jul 1993

What tools or techniques do you think we still need before literate programming is 'ready for the masses'?

We have an incredible proliferation of different representations ("systems") for literate programs and suites of tools for manipulating them. No tool works with multiple representations. To my knowledge, only one representation is designed for easy manipulation by tools. That one has attracted a few tool writers, but none of the more interesting or difficult tools (prettyprinters, hypertext browsers, multi-mode editors) work with it. Many representations support at most a handful of programming languages (a serious flaw) and typesetters (a minor flaw). Many tools are overly complex and perpetuate the problems identified by Ramsey and Marceau (SPE, July 1991). In short, the tools are immature. Progress might be made if we all tried getting behind a single representation and went to work making it support *any* language, *any* typesetter (including WYSIWYG), and if we created some simple but interesting tools, with all the amenities people have grown accustomed to, like automated indexing, prettyprinting, etc. Then again, it might not.

As to techniques, I claim there aren't any. We have Knuth's admonishment to write for human readers, and mine to use lots of peer review, and I think that about exhausts the contenders. Our only book on the subject spends hundreds of pages mostly guiding the reader around the idiosyncracies of the tools. We have exactly two published literate programs of any size, programs which might never have been published if not for the immense prestige their author earned in other endeavors.

Should a literate program resemble a novel, essay, encyclopedia, textbook, or automobile-repair manual? No one knows. Probably none of these---after all, it is a new literary form. But do we create and publish literate programs so that the community can study them, learn from them, and perhaps build an understanding of what a literate program is and how to write one? No! No, we write tools, because that's the only thing we're smart enough to understand. I count myself doubly guilty, since I have written twice as many tool sets as most other contenders.

Well, I apologize for the polemic. Put it down to annoyance. I just finished an article on yet another literate-programming tool, and the article contained neither a clear description of the tool nor any reference to the criteria so clearly set forth by Thimbleby. I wish I knew what the editors and referees were thinking of.

From: **Ian Cargill**
Date: 15 Jul 1993

Well, I apologize for the polemic. Put it down to annoyance.

DON'T! I think this group needs more discussion of this type if we are to advance the 'cause' of literate programming. Since I joined the list, posts have been mostly about the mechanics of specific implementations. I don't say that that is a bad thing, just that it must not be the ONLY thread. I'm a bare beginner, so I not ready to contribute much yet, but think we would all benefit from more discussion of the points that Norman has raised.

finished an article on yet another literate-programming tool

Any good? Reference?

From: **Remo Dentato**

Date: 15 Jul 1993

I agree that this list need more discussions on the concepts of literate programming. Like everyone else here, I presume, I was attracted by the idea of literate programming: what a sense of freedom! The problem, I thought, are the tools: they are too complex. So I started to write my own, simpler, literate tool. During the development I introduced more and more commands, just because could be useful have such commands, so my tool became just a bad copy of the others. Of course I threw away the tool. But it wasn't wasted time: that experience turned to be useful because I understood more about the literate style. Of course I'm just a novice in the literate programming but I think that this new way of thinking is one of the best things that happened to programmers since the advent of full screen editors (:-)).

I think, as before, that the problem is in the tools we use, but just because they are not powerful enough. For example we have to navigate a graph (our webfile) using a linear tool (an editor), it would be much better to have an hypertext tools. The formatting commands and the special commands we use "obscure" the text and the code we are writing, so a WYSIWYG hypertext tool should be even better! Of course we would have to be platform independent (IMO one of the biggest advantages of TeX is its availability for many platforms), and typeset beautiful math and so on.

But here is a problem: more powerful the tools, more powerful the machines we have to use (I simply can't stand to wait too much for, say, the generation of the indices) and I work on various machines that range from IBM RS6000 to MS-DOS (:-( ). For now I have adopted the nuweb system from Preston Briggs, it's clear, fast, language independent, and gives me all I need to be a literate. IMO is the easiest and most natural tool I've used.

The problem is: why using literate programming? Personally I have choose literate programming first of all for helping myself during development, and second for the others that, finally, can read my code and understand what I intended to do. During the development I absolutely don't want to be bothered with typesetting problems (after all maybe none will read my code!) but I still want indexes and cross referencing of macros etc, etc. so I'm currently writing a tool that will do this kind of works for the nuweb system. In conclusion I'm very interested in what you think are the "intimate essence" of literate programming (;-)), both from the experienced literate programmers, and the newcomers.

From: **Michael Haux**
Date: 15 Jul 1993

I have for some time passively followed this discussion list. Literate programming seems to have interesting qualities, leading to better documented and readable code. However, I agree with the recent postings concerning the focus of the list (see Norman Ramsey). The discussion of available tools should also be in the list, but for litprog to be practicable, I think there has to be more. I would appreciate more discussion on themes like: (1) How does literate programming integrate in a SW development process? With software engineering methods? (2) Methods and techniques for literate programming. (3) Litprog feasibility for larger, "real-world" projects.

From: **Norman Ramsey**
Date: 15 Jul 1993

finished an article on yet another literate-programming tool

Any good? Reference?

No. Garbage. Reference withheld to protect the guilty.

From: **Kevin Cousins**
Date: 16 Jul 1993

Fellow Literate People, Recently, Remo Dentato gave an excellent summary of his experience with Literate Programming and opinions. Some background: I work in very much an engineering environment surrounded by DEC & SUN workstations, PCs and MACs. Much of our software development is for PCs to act as controllers for various bits of lab equipment or devices we have built.

The sorts of code that is produced here tends to be extremely cryptic little device drivers (i.e. a couple of hundred lines of source C code) for DOS and WINDOWS :-(It is often difficult to explain all of the intricacies of a given piece of code even to other members of ones own project team. Witness me trying to illustrate code that writes a particular byte to some port in order to toggle specific bits on some home brew board. The illustration is rife with interruptions: Why that bit pattern? Why that port? Why perform that operation on this data? It seems that prolific /* commenting */ is not adequate.

Project management started to come down on us poor code writers demanding better software specification from the programmers! :-o and pleading for better documentation of any code that gets written. (Why programmers are ever writing specifications I'll never understand---in MIS departments, the specification is handed to the programmer who is told "Go away and code something that does this!" True or false?)

It was at this point that Literate Programming came into my life :-) Unfortunately, given the depth of my search for information on the topic, it was not at all obvious how to write my short cryptic routines in such a way as to make them Literate. I have FWEB, am writing in C, use Emacs 18.57 with a web-mode, do not know very much about TeX (but have been through _The_TeX_Book_ once already and discovered there is not much that I have to know to get by on if I don't want anything too fancy), and am following this mailing list with avid interest. (Once again, I wouldn't mind at all if it became a news group: the more people that can read about our experiences, I feel, the faster this technique might mature and turn into something even more wonderful that some people already believe it to be! Seems to me thought that it doesn't matter whether my mail box is full or my news reader overflows---same problem, different windows!)

I have tried WinWordWEB, a nice little tool for those interested in WYSIWYG literate programming. There is no real provision there for hypertext-like coding, however. I feel that implementing a hypertext, WYSIWYG literate programming program using something like HTML could get very cumbersome unless appropriate tools are available to handle the messy repetitive details of the hypertext links. Certainly the idea of a hypertext coding environment has merit: the reason for this post is to broach the subject of Microsoft's Browse Utility which comes as part of their top end compiler packages.

We recently purchased MS C/C++ 7.0. From within their Programmer's WorkBench editor environment , it is possible to generate a 'browse database' containing cross references for any and all definitions and references to source files, data types, indentifiers, etc. present in your code. Relationships between source files and identifier references can be displayed, and almost everything is point and click: see a variable name, want to know where it is defined/referenced? double click on the name, up comes a dialog OR up comes an editor window scrolled to precisely the right point in the file. Which function(s) call(s) this library routine? Point and click!

The ability to have a global view of the code like this is extremely handy. It does absolutely nothing for improving documentation, but when writing code and in particular debugging it, this browse feature is most beneficial. Its a pity that it is solely a MS product, that it must be run on a PC under specific circumstances, but it certainly seems to be able to handle mixed languages (We tried with C and Fortran). I'd like to see this sort of functionality in an X client. The PC stuff from MS gives you this psuedo-hypertext-like stuff through a hierarchy of dialog boxes. Perhaps its not true hypertext , but it sure beats scrolling through hundreds (or thousands) of lines of code looking for what you want!

From: **Eric van Ammers**
Date: 16 Jul 1993

Remo Dentato writes: In conclusion I'm very interested in what do you think are the "intimate essence" of literate programming (;-)), both from the experienced literate programmers, both the newcomers.

I have asked this question several time to the LITPROG netters before. For myself I am getting more and more convinced that the essential quality of litearte programming is to associate a given design step exactly with its consequences in terms of the steps (code) of an algoritm, NOTHING MORE AND NOTHING LESS. In other words, literate programming makes explicit which code lines are responsible for which design decisions

---

From: **Zdenek Wagner**
Date: 16 Jul 1993

I will write a very short message what literate programming brought me. I am not a novice but I do not consider myself an expert either. I had my old programs written about half year ago in CWEB. Each of these 4 programs had about 20 pages of listing. Of course I have forgotten how these progams work inside. Now I had to change these programs in order to alter their functionality. When I counted the changed section, it was about 70% of the program. I managed to do such drastical changes of 4 programs within 3 hours and all programs worked fine after the first compilation without any debugging. I doubt whether I would be able to do the same without literate programming.

From: **Tim Larkin**
Date: 17 Jul 1993

In conclusion I'm very interested in what do you think are the "intimate essence" of literate programming (;-)), both from the experienced literate programmers, both the newcomers.

I believe that WEB and CWEB share one critical characteristic which are missing from some other literate programming systems: the ability to create an exposition in the web which is organized *independently* of the requirements of the compiler. Of course I refer to the infamous modules, which permit me to describe part of function A, then part of function B, then some more of A, followed by some of C, and so forth. Modules free the programmer from having the development of the explanation restricted by linear processing of the compiler. If one cannot break the tyranny of the compiler, one might as well just write comments and make do with a pretty printer. I think this characteristic is close to the "essence" of literate programming, since it allows explanations to follow literate patterns directed toward human readers. I consider it to be more important than tables of contents, indices, or hypertext links.

From: **Osman Buyukisik**
Date: 17 Jul 1993

*I believe that WEB and CWEB share one critical characteristic which are missing from some other literate programming systems: the ability to create an exposition in the web which is organized *independently* of the requirements of the compiler. Of course I refer to the infamous modules, which permit me to describe part of function A, then part of function B, then some more of A, followed by some of C, and so forth. Modules free the programmer from having the development of the explanation restricted by linear processing of the compiler. If one cannot break the tyranny of the compiler, one might as well just write comments and make do with a pretty printer. I think this*

I agree that the modules/scraps are very important, but I think that almost all of the literate programming tools I tried had it! Nuweb, noweb, FunnelWeb, CLiP, WinWordWeb. Without that ability: "to break the tyranny of the compiler", they would be quite useless as some do not even pretty-print the code! With the module concept they help to implement the literate programming paradigm: make it read/look like a book.

From: **David Kastrup**
Date: 17 Jul 1993

Note that the "doc" style option for Literate Programming of TeX macro files does not have order independency or modules. So this is an exception of your list of Literate Programming tools all having sections. However, with TeX macros this problem is alleviated a bit because you usually can define macros in whatever order you like. In addition, TeX scraps in macros are, due to changes of catcodes and other things, much less representable as syntactic units as in other languages with a fixed syntax.

I would, however, strongly recommend using the "doc" style option for doing literate TeX programming. It has the additional advantage that you do NOT need tangle or weave programs, since all typesetting (I believe this might be the case in Noweb as well) appears as comments to TeX when using the style, and TeX is pretty fast in skipping them. This is ok for testing, and for production versions you can use docstrip for getting the comments out. Sorry for this disgression, but I think the "doc" style option is mentioned not often enough, and it really makes TeX programming literate.

From: **Lee Wittenberg**
Date: 18 Jul 1993

Norman Ramsey states (in the middle of a "polemic," most of which I agree with): Should a literate program resemble a novel, essay, encyclopedia, textbook, or automobile-repair manual? No one knows. Probably none of these---after all, it is a new literary form. But do we create and publish literate programs so that the community can study them, learn from them, and perhaps build an understanding of what a literate program is and how to write one? No! No, we write tools, because that's the only thing we're smart enough to understand. I count myself doubly guilty, since I have written twice as many tool sets as most other contenders.

I agree with Norman that we need to start publishing our literate programs. One of the problems, however, is that many editors don't know what to do with them. We need to "educate our editors," to paraphrase Grace Hopper. On the plus side (in the "toot my own horn" department) the _Paradox Informant_ (Jerry Coffey, Elightened Editor) will be publishing a literate program of mine (written using noweb) in the September issue along with another article on _Literate Programming in PAL & ObjectPAL_ ("God willing and the crick don't rise"). I'm also planning on rewriting my Baby Manchester Mark I simulator in CWEB specifically with the idea of publication.

Obviously, I'm not even close to the first to publish a literate program, but it seems that the momentum in the literate programming community has picked up to the point where we need to stop "reinventing the wheel," and start using the wheels we have. [The phrasing is a bit too strong, but the idea is there.] Remember that the literate programming column in CACM died because people were just building tools rather than using existing ones, and that was a good 5 years ago.

From: **Lee Wittenberg**
Date: 18 Jul 1993

I'd like to add something to the discussion of literate programming techniques that is, fortunately, starting to take over from the mail/newsgroup fracas :-). I've noticed that there is a significant difference in _purpose_ between text chunks in webs and "normal" program comments. In a non-literate program, the documentation is there to "comment" on the program source (hence the name). The code itself is definitive. Weinberg's suggestion (Psych. of Comp. Pgmg) that comments be covered during debugging comes from this.

On the other hand, the *text* in a web is what is definitive. I find that most of my debugging consists of making sure that the code chunks agree with their respective text chunks. I would suggest that, when debugging a web, the _code_ sections be covered (at least initially). The names of the code chunks also provide significant information. For example, a recent program of mine included the the following (embarrassingly stupid) bug:

```
@<Do something if |set| is not empty@>=
```

```
if (set == NULL) do_something();
```

Note that the bug is in the code, not the documentation. When the code in a literate program disagrees with its documentation, the fault usually lies in the code, while the opposite is true in traditional programs. This all reflects on what Norman Ramsey said earlier. literate programming is still in its infancy; we've all got a helluva lot to learn. Actually, that's one of the things I like about it.

WYSIWYG webs and debugging

From: **Bryan Oakley**
Date: 15 Jul 1993

Bryan Oakley writes: Some folks have mentioned WYSIWYG webs. At first blush it sounds like a great idea. However, when debugging a web file using a source browser I generally like to see the web source, not the generated .c, etc. source, and I like the compiler to complain by giving line numbers in reference to the orignal web file as well. If the file is WYSIWYG, it seems to me that most (all?) compilers and debuggers would choke on the WYSIWYG stuff. Given all that, is there a FrameMaker web out there somewhere?

Lee Wittenburg writes most eloquently: A WYSIWYG web (at least the one I know about) creates a plain ASCII source file that will certainly not choke any compiler when it is tangled. There is no reason that a WYSIWYG web system couldn't put #line directives (or the equivalent) into the tangled output, as noweb does if a switch is set on the command line. Unfortunately, few languages support a directive to "fake out" the compiler's line counting as C's #line does :-(. Hope this clears up some of the confusion.

Actually, No. I guess I didn't make myself clear. I realize that a WYSIWYG web creates plain ascii files; that much is obvious. However, when it creates the code and inserts the #line declarations, I'm concerned that they will be pretty much useless. If I run a source debugger I would prefer to see the ORIGINAL SOURCE (ie: the WYSIWYG web file), _not_ the tangled code, much like I prefer to see the source code instead of compiled machine or assembly code in traditional software development (ie: I view a tangled source module to be at the same level of abstraction as the ultimate compiled code -- the language file, .c, .f, whatever, is purely a means to an end. However, source debuggers make the assumption that the source is in ASCII, not Word(im)Perfect, Word, Frame, or some other format so it must show the tangled source code. Well, that has no relationship on the real 'source' code, which causes me grief.

I see web systems as truely an improvement when the tangled code is merely a byproduct, and there are sufficient tools to allow one to work with the original source in all phases of development. Why should I code in one language (a web system), then debug in another (a traditional language)? Admittedly, web systems provide great documentation, but for the work I do, content (and useability) is much more important that format. I dare say that I can write equally readable code with just my normal commenting.

From: **Lee Wittenberg**
Date: 18 Jul 1993

Bryan D. Oakley writes: ... text omitted ... when it creates the code and inserts the #line declarations, I'm concerned that they will be pretty much useless. If I run a source debugger I would prefer to see the ORIGINAL SOURCE (ie: the WYSIWYG web file), _not_ the tangled code, much like I prefer to see the source code instead of compiled machine or assembly code in traditional software development (ie: I view a tangled source module to be at the same level of abstraction as the ultimate compiled code -- the language file, .c, .f, whatever, is purely a means to an end. However, source debuggers make the assumption that the source is in ASCII, not Word(im)Perfect, Word, Frame, or some other format so it must show the tangled source code. Well, that has no relationship on the real 'source' code, which causes me grief.

I agree, but _someone_ has to "bell the cat." Would you be willing to write such a source debugger yourself?

I see web systems as truely an improvement when the tangled code is merely a byproduct, and there are sufficient tools to allow one to work with the original source in all phases of development. Why should I code in one language (a web system), then debug in another (a traditional language)?

Again, all it takes is for someone to build the tool.

Admittedly, web systems provide great documentation, but for the work I do, content (and useability) is much more important that format. I dare say that I can write equally readable code with just my normal commenting.

I disagree totally with this last statement. The code I produce using CWEB and noweb are significantly more readable than anything I have ever produced before. I have also had occasion to delve into other people's literate code, and can testify from experience that even mediocre literate code is easier to modify than good non-literate code. Honesty, however, forces me to admit that when I first read about literate programming (in the CACM column), my opinion was "My code is quite `literate' as it is, thank you. I don't need any fancy typesetting tools." I was wrong. I find it more productive to give up my source debuggers in favor of literate programming tools than vice versa. Adding more fuel to the fire.

From: **Marcus Brown**
Date: 19 Jul 1993

Bryan D. Oakley writes: ... text omitted ... Talks about the need for the debugger to work with the WEB source code, not the tangled, unreadable garbage. I see web systems as truely an improvement when the tangled code is merely a byproduct, and there are sufficient tools to allow one to work with the original source in all phases of development. Why should I code in one language (a web system), then debug in another (a traditional language)?

It was a few years ago now, but I was writing CWEB on a Sun workstation. Sun provided a graphical 'dbxtool', which merely put a graphical front end on the 'dbx.' The good thing about it was that it followed the '#line' declarations in the tangled C code, and showed the actual CWEB source code in the 'source code' window. I would think that any tool which followed the '#line' declarations would be able to do this with a minimum of trouble.

One problem I ran into: I had a particular section which (like a good general purpose section should) was plugged into several different sections. That is, I needed to @<Normalize the data@> in several places, so that section was used several times. Trying to debug that section caused a problems: When I placed a break point in the section, 'dbx' wouldn't stop there! I tried it several times, putting breakpoints before and after the section was 'called' ... It just wouldn't stop inside of that section! Finally I set a breakpoint just before the section, then stepped through one line at a time... This worked fine, and I figured out my bug.

The question that remained was: Why wouldn't 'dbx' honor the breakpoint I set in that section? "This exercise is left to the reader..." Just for fun, I'll let you think about it for a while, then give the answer in a few days... Anyway, my experience showed that an 'off-the-shelf' debugger CAN be used to show the WEB source code, with only a few minor problems.

From: **Bryan Oakley**
Date: 19 Jul 1993

Bryan D. Oakley writes: ... text omitted ... Talks about the need for the debugger to work with the WEB source code, not the tangled, unreadable garbage. I see web systems as truely an improvement when the tangled code is merely a byproduct, and there are sufficient tools to allow one to work with the original source in all phases of development. Why should I code in one language (a web system), then debug in another (a traditional language)?

Marcus Brown writes: It was a few years ago now, but I was writing CWEB on a Sun workstation. Sun provided a graphical 'dbxtool', which merely put a graphical front end on the 'dbx.' The good thing about it was that it followed the '#line' declarations in the tangled C code, and showed the actual CWEB source code in the 'source code' window. I would think that any tool which followed the '#line' declarations would be able to do this with a minimum of trouble.

sigh Context is being lost here. I am aware of what the #line directives do, and also what dbxtool does. However, I was refering to WYSIWYG webs. For example, if I web'ed using, say, WordPerfect, dbxtool would have a veritable fit if it tried to display a WordPerfect document in it's source window. Non-WYSIWYG webs I presume would pose no challenge to the average (or at least above average) source code debugger.

{stuff deleted...} Anyway, my experience showed that an 'off-the-shelf' debugger CAN be used to show the WEB source code, with only a few minor problems.

Except, as mentioned, with WYSIWYG webs...

From: **David Kastrup**
Date: 20 Jul 1993

As to source level debugging, and the inability of coping with WYSIWYG literate programming systems: A REAL literate programming debugger would show the lines in the prepared document, not in the WEB source. This could be an interesting challenge for interface design: the source lines had to be \special led into the .dvi-file, and the previewer had to react on messages of the source code debugger and show the appropriate lines.

But with TeX WEBs we have at least the possibility of debugging based on the WEB code. Remember though, that this is not the final solution! And the problems to do "the real thing" with TeX WEBS are about as complicated as with WYSIWYG, only that with TeX WEBs and, say, xdvi, one COULD design a working solution given time and work because the relevant sources (xdvi, gdb, tangle&weave) are all available to the public, and open to discussion, whereas the WYSIWIG approach of the more common editors would entail persuading the producers of commercial systems to build in appropriate support. And, of course, debbuging in the WEBs is a tolerable intermediate solution.

From: **Humberto Ortiz Zuazaga**
Date: 20 Jul 1993

As to source level debugging, and the inability of coping with WYSIWYG literate programming systems: A REAL literate programming debugger would show the lines in the prepared document, not in the WEB source. This could be an interesting challenge

This would be even more difficult than debugging in WYWSIWYG webs, as you can't edit the .dvi (or .ps) file.

From: **Anselm Lingnau**
Date: 20 Jul 1993

David Kastrup said: As to source level debugging, and the inability of coping with WYSIWYG literate programming systems: A REAL literate programming debugger would show the lines in the prepared document, not in the WEB source. This could be an interesting challenge

Humberto Ortiz writes in answer: This would be even more difficult than debugging in WYWSIWYG webs, as you can't edit the .dvi (or .ps) file.

It shouldn't be too difficult just to *show* the lines in the prepared document, given a previewer that is smart enough. I suppose a DVI previewer with a suitable Tcl/Tk interface would come in very helpful here. Of course, then you'd also like to have a debugger and editor which use Tcl/Tk so that they can interact. For instance, if you single-step through a program, the DVI document and the ASCII source will scroll in sync to always display the line that's being executed. But once we're all programming literately, we'll be able to understand all our programs a lot better, and after a while, the code we write will be working right off the bat, so we won't need debuggers anymore. Or will we? :-)

From: **Don Grodecki**
Date: 20 Jul 1993

I would think that any tool which followed the '#line' declarations would be able to do this with a minimum of trouble. ..... Anyway, my experience showed that an 'off-the-shelf' debugger CAN be used to show the WEB source code, with only a few minor problems.

In the same fashion those using WYSIWYG Webs should be able to have an untangled ASCII version of their Webs show up in the debugger. This ASCII file dumped from the Web could be used as the input to "tangle". This would be almost as good as seeing the Web itself, as the debugger source view is usually "read-only" anyway.

From: **Kevin Cousins**
Date: 21 Jul 1993

Don Grodecki writes: In the same fashion those using WYSIWYG Webs should be able to have an untangled ASCII version of their Webs show up in the debugger. This ASCII file dumped from the Web could be used as the input to "tangle". This would be almost as good as seeing the Web itself, as the debugger source view is usually "read-only" anyway.

Here, here! Is there not a utility called 'detex'? I had occasion to use this years ago, so that my thesis could be run through spelling and grammar checkers without the TeX macros appearing. It seems a trivial matter to detex the 'woven' WEB to get a plain ascii version of the final document, which I should expect would be a great way to see the thing in a source level debugger. Any ideas for coordinating '#line's?

From: **Norman Ramsey**
Date: 21 Jul 1993

Here, here! Is there not a utility called 'detex'? I had occasion to use this years ago, so that my thesis could be run through spelling

and grammar checkers without the TeX macros appearing. It seems a trivial matter to detex the 'woven' WEB to get a plain ascii version of the final document, which I should expect would be a great way to see the thing in a source level debugger.

detex is unsuitable for this purpose; look at the output sometime. dvi2tty would be much better.

It would be an interesting exercise to build a literate-program viewer along the lines of Marcus Brown's work but showing the *output* of TeX instead of the (unreadable) input---an essential property in my view. Coordinating such a viewer with a debugger would not be difficult. One approach that would be relatively easy to implement in noweb would be to make a special TeX run in which each documentation chunk appears on a single, arbitrarily large page. The viewer could then display these chunks using standard dvi technology, but could display code chunks in simple ASCII, properly associated with source-line numbers for easy debugging.

A propos of debugging, planting breakpoints with replicated chunks is isomorphic to function inlining, and it breaks most debuggers---but not ldb! (small ad for my dissertation project :-).

From: **Lee Wittenberg**
Date: 31 Jul 1993

It has occurred to me that it should be rather simple (although not trivial) to create a WYSIWYGish symbolic CWEB debugger by wedding a standard debugger to a dvi driver. Some fairly simple changes can be made to the macros in cwebmac.tex -- only \N, \A, and \U need be changed, as best I can figure -- to have these macros insert \specials into the dvi file with section cross-references to be used by the debugger. CWEAVE could also be modified (again, simply but not trivially) to put #line information into \specials, as well. The debugger could use this to establish a correspondence between the .dvi file and whatever debugging info is stored in the object code. It could then generate error message, trace info, etc. geared to the dvi file, allowing debugging to take place completely (or so) with regard to the woven web. It would still be necessary to make fixes to the .w file, but you can't have everything.

A more ambitious debugger might also be able to use the info in the generated index. "Unfortunately, I have neither the time nor the expertise to follow up this project myself, although I would be willing to help in any way I can, if anyone cares to adopt it as their own." -- Thor Heyerdahl

---

Publishing WEB programs

From: **Mary Bos**
Date: 06 Aug 1993

I've been reviewing my litprog mail and came across Lee Wittenberg's comment about publishing literate programs. How about in the literate programming newsgroup, we have literate programming submissions (non-proprietary programs, of course)? For those of us still floundering around in the various WEB's, reading other people's weaves may help us gain a style and code organization. Also, it might help those of use trying to sell management to try WEB to have more examples than what we personally produce. After all, Knuth said, "When was the last time you curled up with a good program to read?"

From: **Anselm Lingnau**
Date: 07 Aug 1993

Mary Bos writes: How about in the literate programming newsgroup, we have literate programming submissions (non-proprietary programs, of course)? For those of us still floundering around in the various WEB's, reading other people's weaves may help us gain a style and code organization.

True. Maybe we should go for comp.sources.literate as well.

From: **Preston Briggs**
Date: 07 Aug 1993

How about in the literate programming newsgroup, we have literate programming submissions (non-proprietary programs, of course)? For those of us still floundering around in the various WEB's, reading other people's weaves may help us gain a style and code organization.

How should we publish? Mailing 20 or 100 page postscript files to the whole group seems excessive. Perhaps a collection could be maintained at an ftp site?

From: **Lee Wittenberg**
Date: 09 Aug 1993

Preston Briggs writes (in response to Mary Bos' original suggestion): How should we publish? Mailing 20 or 100 page postscript files to the whole group seems excessive. Perhaps a collection could be maintained at an ftp site?

Perhaps the .sources newsgroup could be used for announcements and the actual sources (dvi and PostScript) could be kept at a central ftp site. Which brings to mind a problem I've been pondering for a while. For those of us in a "publish or perish" situation, how do we convince our administrators that literate programs made available via ftp (or, indeed, articles in the same situation) are bona fide publications (as I would maintain they are). I've been toying with the idea of using a "Virtual Press" designation, but haven't actually done anything in that vein as yet. Any ideas?

From: **Preston Briggs**
Date: 09 Aug 1993

Lee Wittenberg writes: Perhaps the .sources newsgroup could be used for announcements and the actual sources (dvi and PostScript) could be kept at a central ftp site.

In general, that sounds fine. However, there's some details that should be hammered out. The "source" of a web isn't really posctscript of dvi; it's the CWEB or nuweb or noweb file. I think the typeset output will be fine, though some systems won't make .dvi files (are there any that can't make Postscript)?

Which brings to mind a problem I've been pondering for a while. For those of us in a "publish or perish" situation, how do we convince our administrators that literate programs made available via ftp (or, indeed, articles in the same situation) are bona fide publications

Well, I don't think they really should count as publications. The essence of a "publication", for tenure consideration, is the review process, right? Papers in unrefereed journals are not considered important. Same'll be true for most forms of electronic publication (unless there's a respected reviewing process).

From: **Mary Bos**
Date: 09 Aug 1993

Lee Wittenberg writes: Which brings to mind a problem I've been pondering for a while. For those of us in a "publish or perish" situation, how do we convince our administrators that literate programs made available via ftp (or, indeed, articles in the same situation) are bona fide publications

Preston Briggs responds: Well, I don't think they really should count as publications. The essence of a "publication", for tenure consideration, is the review process, right? Papers in unrefereed journals are not considered important. Same'll be true for most forms of electronic publication (unless there's a respected reviewing process).

Why not have a refereed "Virtual Press" or electronic forum? Reiterating my initial statement, "For those of us still floundering around in the various WEB's, reading other people's weaves may help us gain a style and code organization." Refereeing would help me better appreciate fine workmanship from first efforts - after all engineers learn from elegant solutions to a solved problems, why shouldn't LPer's? There could be two forums perhaps? One unrefereered and and one refereered.

From: **Marcus Brown**
Date: 09 Aug 1993

Lee Wittenberg wrote: For those of us in a "publish or perish" situation, how do we convince our administrators that literate programs made available via ftp (or, indeed, articles in the same situation) are bona fide publications (as I would maintain they are). I've been toying with the idea of using a "Virtual Press" designation, but haven't actually done anything in that vein as yet. Any ideas?

My promotion & tenure committees want 'Refereed Publications' - which means that any publications, whether 'paper' or 'program', would need to be refereed -- In their eyes, if it is not refereed, it is probably trash. Somehow, we would need to set up a jury of referees who would review the submission, pass judgement on its merit, and suggest improvements/additions/... This refereeing is usually anonymous, although some journals also require that at least one member of the 'editorial board' personally endorse a paper

before it is included in the 'Refereed, Meritorious' category.

It may be good to have a repository for any submissions, whether refereed and approved, or not. However, for academic 'publication' credit, there must be a peer review process to certify the relative merit and contribution of each publication. I think that the development of a 'Virtual Press' (as opposed to a physical journal printed on paper) is a great idea, and I heartily endorse thinking and planning about how this could be accomplished. In the beginning, it would need support by some well-known, respected figures to establish the 'academic respectability' of this new form of journal. Unfortunately, I expect that this is more work than the average readership of a mailing list and/or newsgroup would want to volunteer for.

From: **Aaron**
Date: 09 Aug 1993

Marcus Brown said: I think that the development of a 'Virtual Press' (as opposed to a physical journal printed on paper) is a great idea, and I heartily endorse thinking and planning about how this could be accomplished. In the beginning, it would need support by some well-known, respected figures to establish the 'academic respectability' of this new form of journal. Unfortunately, I expect that this is more work than the average readership of a mailing list and/or newsgroup would want to volunteer for.

Some group like ACM or a journal could probably do it best, since they already have the basic review mechanism in place.

From: **Norman Ramsey**
Date: 10 Aug 1993

Which brings to mind a problem I've been pondering for a while. For those of us in a "publish or perish" situation, how do we convince our administrators that literate programs made available via ftp (or, indeed, articles in the same situation) are bona fide publications (as I would maintain they are). I've been toying with the idea of using a "Virtual Press" designation, but haven't actually done anything in that vein as yet. Any ideas?

You don't, because they're not. More than anything else, they are like technical reports, many of which are made available for ftp these days. The problem with ftp is that the audience is unknown (but probably small), and there is no review process. Some publications, like SIGPLAN Notices, require only the concurrence of the editor for publication. Others require peer review. In my experience, peer review improves work enormously. I seldom spend time reading technical reports available by ftp if the same work has been published in a reputable conference or journal. SIGPLAN Notices may be a reasonable place to attempt to publish a literate program. So might Software---Practice & Experience, if the program met SP&E's charter of having something useful to offer to practitioners.

From: **Lee Wittenberg**
Date: 11 Aug 1993

Mary Bos writes: Why not have a refereed "Virtual Press" or electronic forum? Reiterating my initial statement, "For those of us still floundering around in the various WEB's, reading other people's weaves may help us gain a style and code organization."

Sounds like a good idea to me. Although I do not have the organizational skills to run such a forum, I would certainly be willing to serve as a referee. Perhaps Mary would be willing to serve as coordinator, as no actual webbing experience is necessary for the job (although an interest is certainly helpful).

From: **Marcus Speh**
Date: 11 Aug 1993

If there is sufficient interest, you may use the organization of Usenet University for the 'virtual press'. UU has already got a Virtual Campus on the MediaMOO at MIT. I have recently entered UU's board of directors, and we can talk about it (Mary, and who else is interested) any time, maybe privately.

From: **Jim Van Zandt**
Date: 24 Aug 1993

The problem with ftp is that the audience is unknown (but probably small), and there is no review process. Some publications, like SIGPLAN Notices, require only the concurrence of the editor for publication. Others require peer review. In my experience, peer review improves work enormously. I seldom spend time reading technical reports available by ftp if the same work has been published in a reputable conference or journal.

How about publishing in comp.sources.reviewed? Incidently, it's a volunteer effort - have you reviewed something published there recently? (No, I haven't either.)

From: **Dietrich Kappe**
Date: 24 Aug 1993

"I can already envision the appearance of a new journal, to be entitled Webs, for the publication of literate programs; I imagine that it will have a large backlog and a large group of dedicated editors and referees." --- Donald E. Knuth, "Literate Programming (1984)"

Seeing as this once failed (ACM ?) in printed form, perhaps an electronic form, somewhat later in the day, might succeed. I am interested in organizing/editing. I envision the distribution to be in source/tex/postscript form. Any interested parties should contact me by email.

From: **Norman Ramsey**
Date: 24 Aug 1993

I think part of the difficulty in publishing literate programs is that the program itself must be of interest, and it is hard to find interesting programs that warrant treatment at less than book length. The kinds of things I have seen published in the past include algorithms, tools, and systems. Algorithms tend to be so small that any form of presentation works, although I suspect there might be a niche for complete implementations of complex algorithms and data structures. The implementation of tools tends to be mostly of pedagogical interest, and it is more likely to be found in textbooks than in journal articles. The best example is _Software Tools_, which I think is a classic of literate programming, although it predates the coinage of the term.

So systems is where I think the action is. For example, I think Wirth and Gutknecht's book on the Oberon project could have been improved substantially by the use of literate-programming tools. Ditto Holub's _Compiler Design in C_. I can think of a number of "how to cope with DOS/Windows/NT" sorts of books that could benefit from such treatment. Unfortunately my friends and I don't write these books, and I don't know who does. Does anyone out there have experience writing or publishing books containing lots of code?

Inverse comment convention

From: **Dave Love**
Date: 26 Aug 1993

There is at least one other book which is executable like the TeX and METAFONT programs: S.L. Peyton Jones, A D.R. Lester, "Implementing Functional Languages: A Tutorial" (ISBN 0-13-721952-0) which uses TeX and Miranda. The journal TUGboat has lots of literate programming, albeit only in TeX.

I've also seen Haskell articles which are executable, but I don't know if they have been published in journals. [Haskell-like (lazy functional) languages usually support an `inverse comment' convention which is enshrined in the language report. Program source is indicated by leading `> ' and everything else is commentary. Your (literate) program can then be fed either to the Haskell compiler or to, say, LaTeX asis.]

From: **Osman Buyukisik**
Date: 26 Aug 1993

I would not consider the `inverse comment method' literate programming. For one thing you do not have the freedom to write the code-segments in any order you like. If the compiler needs them in a certain order, then you are stuck! However this argument may be mute for functional languages.

From: **Dave Love**
Date: 26 Aug 1993

I would not consider the `inverse comment method' literate programming. For one thing you do not have the freedom to write the code-segments in any order you like. If the compiler needs them in a certain order, then you are stuck! However this argument may be mute for functional languages.

Indeed, in most instances (moot). The same with the LaTeX `doc' system. You might however, have, say, some long boring lexical code and want to relegate part of it to an appendix, as I've seen in one instance. The compiler does need the cases kept together. You could do this with a LaTeX hack to float bits around or elide some inside a comment envirnoment, as I've had occasion to.

From: **Preston Briggs**
Date: 26 Aug 1993

I would not consider the `inverse comment method' literate programming. For one thing you do not have the freedom to write the code-segments in any order you like.

And there's no automatically-generated navigational support (i.e., table of contents, indices, cross references). In fact, is it any better (or different) than ordinary commented source?

From: **John Hamer**
Date: 26 Aug 1993

Osman writes: I would not consider the `inverse comment method' literate programming. For one thing you do not have the freedom to write the code-segments in any order you like.

Preston writes: And there's no automatically-generated navigational support (i.e., table of contents, indices, cross references). In fact, is it any better (or different) than ordinary commented source?

I have used the inverse comment method (aka "Bird tracks") for many (smallish) functional programs, and find it *much* better than ordinary commented source, since I can feed the file through LaTeX (or _whatever_) without change---tangle and weave are no-ops! Having a conventionalised---albeit no-frills---means of writing "executable" reports should not be trivialised. This *is* literate programming, make no mistake. Tools that do code re-ordering are mainly patching up deficiencies in languages like `C; this is not a necessary feature of literate programming. Many visual programming tools provide for (the effect of) code re-ordering, but I would not consider these literate programming tools. Having said that, I must admit to using nuweb for larger programs. The gains are not enormous.

From: **Dave Love**
Date: 27 Aug 1993

Preston Briggs writes: And there's no automatically-generated navigational support (i.e., table of contents, indices, cross references).

LaTeX, for instance, has \tableofcontents, \section, \index etc. and I've used them to generate hypertext documents for navigation around literate programs, albeit not for serious use. You could get variable indices if you cared to generate them with an extra tool and you can use \index; you don't get the index from the current noweb either.

In fact, is it any better (or different) than ordinary commented source?

IMHO, yes, although based on experience with TeX and Lisp/Scheme rather than Haskell. (You can hack your Lisp reader to obey conventions about the input format.) For instance, the difference between the (extensive) comments in the source of LaTeX and the add-ons documented with the doc option is considerable, I think. The inverse commenting reflects the emphasis on the commentary rather than the code. It marks what's code and what's commentary if you want to be cleverer with extra tools, and there's no overhead if you don't. I can't imagine a book of normal commented code c.f. the Peyton Jones/Lester one. I think what you get from the typography is considerable, especially if you've got mathematics to present in the commentary.

Code reordering

From: **Lee Wittenberg**
Date: 27 Aug 1993

John Hamer writes: .... Tools that do code re-ordering are mainly patching up deficiencies in languages like `C; this is not a necessary feature of literate programming. Many visual programming tools provide for (the effect of) code re-ordering, but I would not consider these literate programming tools.

This is probably a bit of a flame, but I think I ought to say it anyway. IMHO the thing that makes literate programming "work" is the ability to have a program [in Djikstra's words] "written down as I can understand it, I want it written down as I would like to explain it to someone." Code re-ordering is what allows this. John's point about visual programming tools is well taken, though. In many ways the literate programming and VP philosophies are diametrically opposed. I would suggest that code re-ordering is a necessary, but not sufficient feature of literate programming.

From: **Mike Yoder**
Date: 27 Aug 1993

Lee Wittenberg, responding to John Hamer, wrote: I would suggest that code re-ordering is a necessary, but not sufficient feature of literate programming.

Although this sounds plausible, it is unsupported by empirical evidence. My tool does not reorder code; this seemed to be a serious obstacle precisely once. On that occasion, I just sent the various parts to separate files and catenated them together in the build. The same holds for macro processing: it isn't part of the tool, and this tool was created by adding features as they became needed. If you need macro processing, you can write your code in a macro language like m1; this is simple because the tool is independent of source language. I strongly believe literate programming tools should not have intrinsic language dependencies; what is the point of hobbling yourself this way? At a minimum, most of the literate programming texts I write emit both source code and UNIX makefiles. Frequently, they emit shell, awk, or sed scripts as well, and sometimes it is necessary to include C or assembly code in a program that is mainly written in (say) Ada or Pascal. What is done well by other tools need not be part of a literate programming processor. Code re-ordering can be useful, but it is by no means necessary.

From: **Michael Koopman**
Date: 27 Aug 1993

Michael Yoder wrote in High Level Language: Although this sounds plausible, it is unsupported by empirical evidence. My tool does not reorder code; this seemed to be a serious obstacle precisely once. On that occasion, I just sent the various parts to separate files and catenated them together in the build.

As a newbie I find high level language sensitivity and code reordering a great benefit. Code reordering by the web processor may not be necessary for literate programming but it seems much easier than your alternatives. Certainly we must agree that the order of code which makes the compilers happy is unlikely to be the same as that which allows the literate programmer to enlighten her audience.

What is done well by other tools need not be part of a literate programming processor. Code re-ordering can be useful, but it is by no means necessary.

Tools that can't parse code would have difficulty providing the pretty printing capability which greatly enhances the appearance and improves readability of the code scraps, IMNHO. To perform the pretty printing and not provide for code reordering seems like going out in the rain with an umbrella full of holes. To add many macro executives or file creating directives seems like a lot more work than the programmer should like to manage, especially in the light of relatively simple techniques for code reordering with tools like FWEB.

From: **Osman Buyukisik**
Date: 29 Aug 1993

It seems like some people use literate programming to pretty print/document the code. I think it can be very useful in the design phase as a PDL if the literate programming tool can reorder code. To some just documenting (even with the aid of TeX) seems to be enough to be an literate programming tool. I disagree. It also has to be language independent as sed, make and other `languages' are used often, and reorder code, files, and has a minumum indexing feature. However, this area seems to be wide open to personal feelings.

From: **Mike Yoder**
Date: 31 Aug 1993

Greetings to Michael Koopman and other literate programmers. Your response in some measure fits into a very common pattern in discussions: you have stated true statements which don't contradict what I said. You find code reordering to be useful: I can't contradict that. I find it unnecessary; presumably you don't disagree with that either. However, there are some statements I do disagree with, for example:

Certainly we must agree that the order of code which makes the compilers happy is unlikely to be the same as that which allows the literate programmer to enlighten her audience.

No: I do not agree. There is not a single best way of enlightening the audience, as your phrase "the same as" would suggest, but many. There are also many orders for presenting the code in a way that makes the compiler happy, in the common case where it is a package or module being presented rather than a program. (Perhaps because of my various programming biases, including literate programming, my programs tend to be made of many modules and a small "main" program.) In the languages I use (mostly Ada and extended Pascal), I find it easy to present the code in an order also useful for exposition. It is particularly easy with Ada, which allows subunits.

To add many macro executives or file creating directives seems like a lot more work than the programmer should like to manage, especially in the light of relatively simple techniques for code reordering with tools like FWEB.

This would be true if it applied, but one directive in nine years of using the tool hardly counts as "many." The directive takes up 7 lines in a make file: not a significant amount of extra work for a nine year span. It is absolutely certain (to me, anyway) that the amount of extra work caused by the absence of the reordering ability has been far less than the work it would have taken to implement a macro capability. (If there were enough users of the tool, this would presumably change, but it isn't clear where the crossover point is.)

Finally, you opined that pretty printing was an important capability. Again I disagree: my pretty printing is (in effect) done by Emacs as the code is put onto the electronic page, so I consider this a feature duplicating the capabilities of other tools. And I would ten thousand times rather have language independence than pretty printing, even aside from the fact that my personal experience with "pretty" printing is that it often comes out ugly.

Lest I seem too negative, let me say that I believe a macro capability to be useful enough that a maximally good literate programming tool "ought" to have it. But the gains are more political than practical: there are enough people who won't touch a literate program tool that lacks it that one lacking it can't succeed even if it were technically quite sufficient. Put another way: the real gains are small, but the perceived gains are large. But by no means do I mean to imply that the gains are zero.

From: **Christian Lynbech**
Date: 31 Aug 1993

Mike Yoder writes: No: I do not agree. There is not a single best way of enlightening the audience, as your phrase "the same as" would suggest, but many. There are also many orders for presenting the code in a way that makes the compiler happy, in the common case where it is a package or module being presented rather than a program. (Perhaps because of my various programming biases, including literate programming, my programs tend to be made of many modules and a small "main" program.) In the languages I use (mostly Ada and extended Pascal), I find it easy to present the code in an order also useful for exposition. It is particularly easy with Ada, which allows subunits.

I'm sorry if I'm going back on something already discussed. I do not think that code reordering, in itself, is the true virtue of literate programming. Too many languages have sufficient `presentation freedom' to keep both the compiler and the audience (somewhat) happy. It is more the ability to interleave explanation and (totally order indepedent) code, in manageable chunks and at the power of report formatting/production. A nicely formatted report (or the like) with a something like a table of contents, nested sections and cross references (you were talking about FWEB, weren't you?), is far superior in my view. But it is not the ability to start with the main procedure, but rather the ability to have a properly formatted piece of text, explainig the finer points, that makes literate programming take off. I have yet to see any program so thoroughly documented that it rivals any of the (admitted somewhat few) literate programs with regards to readability and understandability.

But remember, the issue is not whether it is *possible* at all, to program without literate programming in this or that form. The majority of the world seems to be doing reasonably fine without it.

From: **Tony Coates**
Date: 31 Aug 1993

On the subject of the value of code reordering, I have to say that even if it isn't *necessary*, for the C++ code that I write (using FunnelWeb), I find it invaluable to put the definitions and implementations of the methods beside one another in the FunnelWeb source, although they end up in separate files (.h and .cc) afterwards. This may not be necessary, but if one of the goals of literate programming is to reduce programmer errors, then having the definitions side by side with the implementation certainly helps me. The errors usually aren't serious, in the sense that the compiler tells me of any mismatch, but I just save a lot of time when the two are together, because I can see instantly that the both declarations match. Yes, with Emacs I can look at both a .h and a .cc file at once, but I just don't find it as convenient as having things together in one file.

So, for me, a LitProg tool that couldn't reorder code and generate multiple source files would not be nearly so useful as what I have now. You can argue whether I *really* save so much time, but I have to say that having related declarations together in one file is much easier for my poor little brain than having them spread across two or more files.

From: **Mike Yoder**
Date: 31 Aug 1993

Tony Coates writes: So, for me, a LitProg tool that couldn't reorder code and generate multiple source files would not be nearly so useful as what I have now.

I may have created a false impression with my earlier postings. My tool makes multiple source files, but doesn't reorder code within them. Its processing of the files is mostly limited to conditional inclusion, e.g. "@case host" and "@case target" where 'host' and 'target' are user-defined enumeration types.

From: **Lee Wittenberg**
Date: 31 Aug 1993

Michael Yoder brings up an interesting point: Lest I seem too negative, let me say that I believe a macro capability to be useful enough that a maximally good literate programming tool "ought" to have it. But the gains are more political than practical: there are enough people who won't touch a literate program tool that lacks it that one lacking it can't succeed even if it were technically quite sufficient. Put another way: the real gains are small, but the perceived gains are large. But by no means do I mean to imply that the gains are zero.

The interesting thing is that code reordering *is* macro expansion, at least in the literate programming systems I am familar with. To be more precise, it is macro expansion without parameters (the possibility of allowing parameters in chunk names generated a lot a traffic a while ago, so I won't delve into that again). A rose by any other name?

From: **Michael Koopman**
Date: 31 Aug 1993

Christian Lynbech writes: Certainly we must agree that the order of code which makes the compilers happy is unlikely to be the same as that which allows the literate programmer to enlighten her audience.

Mike Yoder writes: No: I do not agree. There is not a single best way of enlightening the audience, as your phrase "the same as" would suggest, but many. There are also many orders for presenting the code in a way that makes the compiler happy, in the common case where it is a package or module being presented rather than a program.

I concur that there is not a single best exposition for any communication involving a literate program (WEB) or it's artifact (WEAVE). However, independence from order restrictions imposed by the compilers is significant to the use of literate programming tools in the maintenance of existing large implementations. Unfortunately I must work with a legacy of non-highly order independent programs, e.g., C, Fortran. Of course, burning all this Neanderthal code and restarting from scratch with *modern* tools is the Right Thing. Chipping away with WEB based exposition regarding the algorithms and implementation strategies has low utility. Rewrites where necessary to attain the elegance of right thinking post-modern Application Interfaces is kludgy beyond acceptable limits, except for the minor requirement of delivery.

Anyone have a foolproof software paradigm conformal mapping implementation? A tool to test if a software paradigm is defined on an orthogonal basis would help, too.

I do not think that code reordering, in itself, is the true virtue of literate programming. To many languages has sufficiently `presentation freedom' to keep both the compiler and the audience (somewhat) happy.

Turning the NIH (not invented here) volume to maximum and rewriting all existing code in, admittedly, highly improved implementations, such as OOA/D and ADA 9x, would be marvelous and the code order restrictions of the compiler could be reduced to acceptable limits. This approach may not be possible for many.

It is more the ability to interleave explanation and (totally order indepedent) code, in manageable chunks and at the power of report formatting/production. A nicely formatted report (or the like) with a something like a table of contents, nested sections and cross references (you were talking about FWEB, weren't you?), is far superior in my view.

A brief retrace to pretty printing. Language sensitivity is a boon to the programmer in automagically generating an index and cross references. Language independence is valuable, also. The two are not mutually exclusive. However, the complexity of the literate programming tool that generates index entries by parsing code is greater than one which requires the author's interjection of 'escaped sequences' to accomplish the same. In this same breath, the escaped sequences are hazardous to the health of language independence.

A WEB should not be a static document, i.e., equivalent to it's current WEAVE. If I need to create an exposition with a focus different

than the current WEAVE, or need to present the same WEB to a different audience, I want to be able to move, splice, dice, fold, spin and mutilate the current WEAVE and still end up with a compilable TANGLE. I loose all my impact if the WEB breaks and can no longer do the stupid dog trick of the compiled TANGLE. The goal of a coherent exposition (logical WEAVE) should eliminate much of the pathology associated with arbitrarily moving pieces of code scraps about in the WEB. (I seem to have abstracted beyond a valuable thesis; hypothetical premise).

*But remember, the issue is not whether it is *possible* at all, to program without literate programming in this or that form. The majority of the world seems to be doing reasonably fine without it.*

Drat, the Real World, again. FOO.

Little TeX needed

From: **Uttam Narsu**
Date: 27 Aug 1993

I taken a look at some examples of literate programming (Knuth's TeX book, and the K&R samples on 192.92.115.8) and I was wondering if one has to learn TeX to be able to do literate programming. It seems that many of the macros one wants to use can be hidden through the selection of an appropriate view in an extensible editor. Has anyone implemented a multi-window literate programming editor? (I just don't see our programmers being very happy about learning TeX in order to do literate programming.)

From: **Bart Childs**
Date: 27 Aug 1993

I have looked at several dozen WEBs from a wide array of programmers. The sad truth is that nearly everybody that use WEBs will put in almost no TeX commands on their own. I say that is sad because just a few can be a significant help. A user can buy Art Samuels' "First Grade TeX" from the TeX users group for less than $10. It is 40 pages long and has more TeX than is used in 99% of the WEBs written by 95% of the users.

Donald Knuth, Silvio Levy, and John Krommes are not in that majority. Some of the WEBs I have written use more TeX, but most don't. The most common exception to this is that I often include lists (item type commands) in documentation portions of the code. In these I will nest these [bgroup -- egroup]s and change the parindent and parskip to format them in a manner that is more suitable to my tastes. Our web-mode for emacs has almost no TeX support because it is not really needed. The AUC-TeX from Denmark has a great amount of TeX support and we might incorporate a little of it in the future.

I was visiting with Ross Williams (the author of Funnel WEB) last month. I pointed out that I thought the lack of HLL sensitivity and therefore the lack of index features was a great lacking of his contribution. He readily acknowledged that Funnel WEB had short-comings but made an eloquent statement that I will try to do justice to. It was something like this: "The great contribution of the WEB style of programming is that you can organize the code in the same way you think of it. I outline the code in a logical way and supplement that with documentation as I need. When I want to finish the details of a part, it is easy with the guidance I have provided and I simply do it. I rarely make a printed version of the code." I offer Ross my apologies for any inaccuracies.

I strongly believe that `the use of TeX is an excuse (and not a reason) to avoid literate programming.' A one-day training course in TeX is far more than is needed and gives more formatting capabilities than is even possible with most WYSIWYG systems. TeX could be made invisible to most literate programming users.

From: **David Kastrup**
Date: 27 Aug 1993

Uttam Narsu writes: I taken a look at some examples of literate programming (Knuth's TeX book, and the K&R samples on 192.92.115.8) and I was wondering if one has to learn TeX to be able to do literate programming. It seems that many of the macros one wants to use can be hidden through the selection of an appropriate view in an extensible editor. Has anyone implemented a multi-window literate programming editor? (I just don't see our programmers being very happy about learning TeX in order to do literate programming.)

W E L L, strictly speaking the use of CWEB/WEB using the appropriate tangle/weave programs is straightforward, not needing any knowledge of TeX. The macros and constructs are particular to WEB, but the front end does not really require much if any knowledge of TeX (although you may not use characters like $&^#{} etc.). However, at least a knowledge of TeX formulas is to be heavily recommended, because one of the WEB advantages is that it is possible to include mathematical basics of algorithms in a readable form (And I do not consider forms like sin((2*PI*N/180.0)+phi)*xyz)*cos((2*PI*N/180.0)+theta) a particularly readable form). And of course, the inclusion of tables is a real pain in the neck. If the current version of CWEB runs with LaTeX, than this at least is simplified.

I agree, however, that the better you make the documentation part, the more of TeX resp LaTeX resp. whatever formatter is used, shines through. The current WEBs are fixated mainly on TeX because:

- Knuth sort of started the whole biz
- They are ASCII input, good typeset output
- They are freely available, so that the fixation does not seem as severe restrictive as orientation on a certain commercial product would seem.

It would be easily possible to adapt those WEBs to other typesetting software, however, such advances would make the approach less portable probably. There are degrees, however. For instance, I think that a Wordperfect WEB would suffer from serious portability problems, although not necessarily from acceptance problems (although, I admit, I am TeX-spoiled and do not want to learn some stupid WYSIWYG system with slightly better than typewriter appearance. This is a topic for other fruitless "I have better SW than you" discussions, however).

Basically, however, one could design sort of a "generic" WEB, which would, through the use of text processing definition files, produce output for dedicated word processors. I am afraid, however, that pagination and indexing would have to be included into the WEBs because much of the work here in current WEBs is done by TeX (and why not?). With a bit of discipline, sources could be kept somewhat formatter- independent, although leaving out formulas etc would seem too harsh a restriction for me. But better a literate program without nice formulas than none at all!

So what's the point? Maybe you should try selling your programmer's TeX through the back door, show them that a WEB need not contain much TeX knowledge. Have at least one TeX guy ready to help doing formulas etc. Support the development of formatter-independent WEBs. BTW, LaTeX is not that hard to start with, and I think that CWEB 3.xx will support LaTeX as well (although I am not that sure).

I am pretty sure, however, that it would be preposterous to demand that all of your programmers should learn mastering plain TeX. This would be madness. But LaTeX should be on the tolerable side. It is unfortunate that there are not that much WEB systems around which support other word processors. Thus you are forced promoting literate programming and TeX at the same time, a bit on the heavy side.

From: **Preston Briggs**
Date: 27 Aug 1993

I was wondering if one has to learn TeX to be able to do literate programming. It seems that many of the macros one wants to use can be hidden through the selection of an appropriate view in an extensible editor. Has anyone implemented a multi-window literate programming editor? (I just don't see our programmers being very happy about learning TeX in order to do literate programming.)

One of the central features of literate programming is high-quality output. Most systems use Tex (or Latex) though there are some now that use Word and older approachs using troff. I think the use of macros (scraps, modules, ...) is orthogonal to the use of Tex. A fancy editor is nice, but not essential.

About learning Tex, ... maybe there's 2 kinds of programmers: 'philes and 'phobes. The 'phobes aren't going to like learning anything new, but the 'philes will enjoy it. Lots of us really _like_ being able, finally, after all these years, to make our programs look nice! I'd try and hire more of the enthusiastic sort and weed out the others.

From: **Lee Wittenberg**
Date: 27 Aug 1993

David Kastrup says: For instance, I think that a Wordperfect WEB would suffer from serious portability problems, although not necessarily from acceptance problems (although, I admit, I am TeX-spoiled and do not want to learn some stupid WYSIWYG system with slightly better than typewriter appearance. This is a topic for other fruitless "I have better SW than you" discussions, however).

I don't know about Wordperfect, but the WinWordWEB system certainly suffers from these deficiencies. Although the output is much better than "typewriter appearance," I find it much easier to use noweb with LaTeX, for the same results.

From: **Uttam Narsu**
Date: 27 Aug 1993

I guess I was less than clear in my original message; I'm not terribly interested in a WYSIWYG system, nor am I averse to learning TeX (or the WEB constructs) (after all it's just another language), but I am interested in leveraging knowledge that I or my colleagues already possess.

We currently use EMACS for most of our editing or source (much of the maintenance of our C++ coding standards is handled through emacs macros). So I suppose an emacs package (which I understand is available with FWEB) would probably do just fine. As a finite element analysis shop, just the ability to document/typeset equations with the code would be tremendously useful.

Preston Briggs wrote: About learning Tex, ... maybe there's 2 kinds of programmers: 'philes and 'phobes. The 'phobes aren't going to like learning anything new, but the 'philes will enjoy it. Lots of us really _like_ being able, finally, after all these years, to make our programs look nice! I'd try and hire more of the enthusiastic sort and weed out the others.

And then there are those of us who love learning new things, but honestly don't have the time. That is the reality of life in a commercial concern with schedules and managers deciding whether we are phobes or philes.

From: **Stephen Fulling**
Date: 27 Aug 1993

As a finite element analysis shop, just the ability to document/typeset equations with the code would be tremendously useful.

To repeat what others have said (very well), (1) This is precisely the place where TeX is most valuable. (Accept no substitutes!-) (2) The basic TeX needed to set equations is easy to learn, and most of the WEB users do not need to know more TeX, provided that there are a few people around to help out in difficult situations and you are prepared to live with occasional infelicities of spacing, etc.

From: **Marcus Speh**
Date: 27 Aug 1993

[[copied from the HyperText info on LitProg which I am maintaining-- for more, see
ftp.desy.de:/pub/www/projects/Announce/LitProg.txt]]

```
Editing WEB files with the GNU Emacs editor
*******************************************
```

If you are developing your WEB, CWEB or FWEB programs using the GNU Emacs editor, there is web-mode.el by Mark Motl <motl@cs.tamu.edu>; the corresponding GNU Emacs mode can deal with WEB, CWEB and FWEB.

It is capable of many things, including jumping to sections and modules, inserting (and previewing) index entries, hiding and exibiting the body of a .web file (showing the tree), inserting, quoting, and consistently renaming modules etc. It supports change files and journal files. It is especially useful when dealing with large .web files not to have to deal with monolithic files.

Detailed information is contained in the User's Manual (PostScript). Here is a reference card (PostScript). The sources can be retrieved from here. The latest version should always be available via anonymous FTP from ftp.cs.tamu.edu, or in Europe from ftp.th-darmstadt.de as web-mode.tar.Z [in directory /pub/programming/literate-programming/Tools]

From: **Christian Lynbech**
Date: 30 Aug 1993

Let me just repeat what many people has already said. Learning the basic use of TeX or LaTeX is not very complicated. I only really know LaTeX, but the basics are in fact quite easy. I claim that a few hours of reading/training, and a copy of the refcard, you can start produce nice looking documents. Beware, however, that disagreeing with the decisions made by TeX/LaTeX, in spacing or placement of things like figures, may need considerable experience, if you want to have your way. Too many newcomers are frustrated by this, in my experience at least. I, as a happy LaTeX'er, find this a feature rather than a problem, far outweighed by the power.

Alternatively, you may want to check out the FunnelWEB package. I haven't looked much into it myself, but I vaguely remember from my browsing through the manual, that it sort of supports formatting, in the sense that it provides a simple set of formatting commands. The nice part is that it can get you started formatting very quick (and in a TeX indepedent way). The bad part is that you are likely to find the possibilities rather limited. But if your requirements aren't fancy, perhaps this would be something for you.

From: **Lee Wittenberg**
Date: 30 Aug 1993

Bart Childs writes: I have looked at several dozen WEBs from a wide array of programmers. The sad truth is that nearly everybody that use WEBs will put in almost no TeX commands on their own. I say that is sad because just a few can be a significant help.

I can support this as well. The webs my colleagues at Tipton Cole + Co. are writing use a minimal amount, mostly from a template I provided that includes things like PVCS support, tables of contents, etc. Whenever I get a chance, I try to point out an interesting TeX/LaTeXism that might be helpful. I also try to exercise TeX a bit more than necessary in my webs to take advantage of the "Gee whiz, can I do that, too?" factor.

A user can buy Art Samuels' "First Grade TeX" from the TeX users group for less than $10. It is 40 pages long and has more TeX than is used in 99% of the WEBs written by 95% of the users.

I would also add Michael Doob's excellent _A Gentle Introduction to TeX_ to this list (sic). It's freely available over the Internet; I got

my copy from pip.shsu.edu.

I strongly believe that `the use of TeX is an excuse (and not a reason) to avoid literate programming.' A one-day training course in TeX is far more than is needed and gives more formatting capabilities than is even possible with most WYSIWYG systems. TeX could be made invisible to most literate programming users.

Again, I agree completely. The necessary TeX/LaTeX commands (for noweb use, at least) take less than a page -- less than half a page, actually, since this list includes the \section commands, which are not _absolutely_ necessary (but incredibly useful). I've found that it only takes about an hour to teach someone the basics.

From: **Mike Yoder**
Date: 31 Aug 1993

To add a single datapoint to the pool: my tool doesn't need TeX; it has three output modes, one of which produces LaTeX. It has a mode in which it generates LN03 output directly which is only used in special situations and when desperate; the third mode generates Interleaf(tm) ASCII form, and the output is of quality comparable to the quality of the LaTeX output.

The mode which makes the Interleaf form was commissioned by Apollo Computer (before it became part of HP): this is because they had standardized on Interleaf for documentation.

Opinion: an literate programming tool ought to be mostly formatter independent but allow for escape sequences so formatter commands can be included directly. I often use these to insert Tex or LaTeX sequences, and it is usually simple to arrange that the commands are ignored in other modes so the only effect is to have the output be prettier in TeX mode than the others. Occasionally I assume that the document will only be printed in the TeX mode.

Literate programming with troff or texinfo

From: **Norman Ramsey**
Date: 27 Aug 1993

There's been much discussion of whether TeX is essential to literate programming, and if so, why. Ignoring the argument that TeX is used because it produces superior documents, I think there are two reasons: one sociological and one technical (and partly sociological). The sociological reason is that literate programming has achieved what credibility it has primarily because of the enthusiastic support of Don Knuth. People who are willing to follow Don out on a limb to practice literate programming are also likely to follow him in the use of TeX---after all, there is a much stronger consensus about the value of TeX in its arena.

The technical reason is that all literate-programming tools have to manipulate an underlying representation of a document to produce code. Of lesser stature are the non-reordering tools like doc.sty and cnoweb, which manipulate code to produce a docuement. In both cases life is infinitely easier if the underlying representation is well documented, and preferably ASCII. (The only counterexample I know is Lee Wittenberg's WinWordWeb, which I am eager to hear more about.) Before TeX83, there were quite a few of these things around: Scribe, troff, Waterloo/IBM Script. (I don't count gml since that was and is a markup language, not a formatter.) Scribe is dead (?), and heaven knows what goes on in the IBM world, but troff is certainly alive and well in large parts of the old Bell organization.

So why don't literate programmers use troff? They used to---one of the first non-Knuth tools was Harold Thimbleby's CWEB with troff. One answer is that many literate programmers are young people---the young are always more willing to try new things---and I don't think many young people are learning troff these days. TeX is too much better. Another is that I've posed the question wrong; it should be "why don't troff users write literate programs?" Why, because they have no tools. It's bad enough to try something new, but it's worse if you have to jettison your years of troff experience.

I have a standing offer open to any troff user interested in literate programming: if you will show me how to get the stuff to look right on the page, and if you promise to write at least one literate program, I will write a troff back end for noweb. Tell your friends.

From: **Sean Boyle**
Date: 30 Aug 1993

So why don't literate programmers use troff? They used to---one of the first non-Knuth tools was Harold Thimbleby's CWEB with troff. One answer is that many literate programmers are young people---the young are always more willing to try new things---and I don't think many young people are learning troff these days. TeX is too much better. Another is that I've posed the question wrong; it should be "why don't troff users write literate programs?" Why, because they have no tools. It's bad enough to try something new, but it's worse if you have to jettison your years of troff experience.

To be fair, it should be pointed out that [nt]roff has one strength over TeX, the ability to produce straight text output. I am often required to send out a document for review via Email. With TeX, the only choices I have are sending out the source file, the DVI file

(uuencoded) or PostScript. Any of the above choices are clearly unacceptable to most of my peers and management. Yes, I know about dvi2tty and such utilities. Theoutput fromthese isunacceptable, probably dueto kerningit lookslike this... Without a *lot* of mucking around with the text, it looks terrible. Nroff does a fair job of coming up with a straight text representation. BTW, I still use TeX (until my manager kills me).

From: **Michael Koopman**
Date: 30 Aug 1993

Sean Boyle wrote to the good readers of LitProg: To be fair, it should be pointed out that [nt]roff has one strength over TeX, the ability to produce straight text output. I am often required to send out a document for review via Email. With TeX, the only choices I have are sending out the source file, the DVI file (uuencoded) or PostScript.

Does some TeX hack know of a good detex utility? I have discovered the utilities described in the following excerpt from the README file found in the examples directory under ftp/cweb on anon ftp at labrea.stanford.edu. Their purpose is not to remove TeX and leave a semblance of formatting, though.

"The extex and wordtest programs, by Don Knuth, are useful spellcheckers. After "make extex.run" say also "ln extex excweb"; this gives you two filter programs, extex and excweb, that remove extraneous material from TeX and CWEB source files, respectively. To use them, you can say for example cat foo.w | excweb | spell (or replace "cat foo.w" with "wmerge foo.w bar.ch"). A similar pipeline for TeX files would be cat foo.tex | extex | spell Even better is to replace "spell" by "wordtest /usr/local/lib/dict/words", if you have a suitable dictionary. One such dictionary in the public domain can be found in directory ~ftp/pub/dict at labrea.stanford.edu, available via anonymous ftp. To make your own dictionary from given ones, cat dict1 dict2 dict3 dict4 | wordtest > words is quick and effective. See the documentation of wordtest for more info."

Norman Ramsey writes: So why don't literate programmers use troff? ... It's bad enough to try something new, but it's worse if you have to jettison your years of troff experience.

Know of no help with this, except, doesn't all that [nt]roff knowledge easily map over to the TeX domain? Perhaps you need the wetware upgrade? ;-)

From: **Bart Childs**
Date: 30 Aug 1993

Re: Thimbleby's paper on CWEB using C and troff. It is not worth my time to go back and read it carefully, but I recall a point made in that paper is that the limitations of troff were one of the most significant problems in creating the system. If having something that can produce a screen oriented version is that important, doesn't texinfo do that to some extent? I would rather work on the future rather than ... TeX and laser printers are so reasonably priced that I wonder what the gain is?

From: **Christian Lynbech**
Date: 31 Aug 1993

Texinfo should do the trick on straight text output. There is even the added feature of being able to produce an info file, i.e. a hypertext representation of the program! I have had this idea in the back of my head for quite some time, but I haven't had time to investigate it much. Certainly, producing a *usefull* program.info file, takes some care and consideration.

I am not much of a texinfo hacker myself, but since texinfo is a juiced down version of TeX (sort of), wouldn't that also solve some of the learning problems reported by people. Some of the principle behind texinfo (as I understand it), is exactly to provide those macros you need and use, rather than the full power of TeX, enabling a precise and nice looking ascii equivalent. There is also a standalone version of texinfo (available from GNU), so you can use it without having neither TeX nor emacs.

Incidently, there is also a LaTeXinfo package, for those already into LaTeX. I have had a chance to look into it, and it feels pretty complete, i.e. it has most of the stuff I tend to be using all the time, including some list making environments. More important perhaps, it looks as ordinary LaTeX, you write \section rather than @section and \begin{description}\end{description} rather than whatever the Texinfo equivalent looks like. The downside is that its maintainance situation is a bit in the dark. There was a discussion on gnu.misc.discuss about how to reach the author, and apparently nobody had succeeded. But the package works pretty well as it stands. I've got a copy of what appeared to be the latest version at the time, if anybody is interested.

From: **Sven Utcke**
Date: 31 Aug 1993

Bart Childs wrote: If having something that can produce a screen oriented version is that important, doesn't texinfo do that to some extent?

Ghee, *what* is texinfo?

From: **Marcus Speh**
Date: 31 Aug 1993

Since HyperText is the talk of town again--what happened to Stephen Cross' project on hypertextified literate programming?

The downside is that its maintainance situation is a bit in the dark. There was a discussion on gnu.misc.discuss about how to reach the author, and apparently nobody had succeeded.

I did not follow the dicussion on gnu.misc.discuss. I have very good experiences with Texinfo maintenance, though. The relevant guy is Roland McGrath (FSF), roland@gnu.ai.mit.edu. He is very responsive and gets into contact with the Texinfo author(s) fast. Incidentally, I agree with all of Christian's conclusions.

*Sven Utcke writes: Ghee, *what* is texinfo?*

[repost from Dec 92] Texinfo is a documentation system that uses a single source file to produce both on-line information [preferably using the GNU Emacs info reader] and printed output. Texinfo does not require the GNU Emacs editor -- it's just much simpler if you have one [the same is true for the Info facility -- there is a standalone "info" executable]. The current distribution [texinfo-2.1?.tar.Z] can be retrieved via Anonymous FTP prep.ai.mit.edu in directory pub/gnu. Installation is easy since the FSF [Free Software Foundation] provides self-configuring files for a wide variety of machines. Without Emacs, what the minimum needed is two executables,

```
o "texi2dvi" for the step .texinfo -> .dvi, and
o "makeinfo" for the step .texinfo -> .info (i.e. text),together with a collection of macros
o "texinfo.tex".
```

If you have Emacs on your system, most probably you will also have those files somewhere in a /local/ dir. There is also the texi2roff formatter which makes you independent of TeX available from prep.ai.mit.edu.

From: **Christian Lynbech**
Date: 31 Aug 1993

Unfortunately, the LaTeXinfo package was not created by the GNU folks. My manual (dated june 17, 1992, version 1.7) states Michael Clarkson as author (in addition to R. Stallman and R. Chassell, but that must refer to the original texinfo package). I quote here another mail I got on the subject, for any interested parties.

Our homebrew "literate programming" system is a Texinfo-based system (actually) that tries to look like LaTeX. I posted a note about it a while back. You'll find all the stuff on ftp.dcs.glasgow.ac.uk, pub/haskell/glasgow/lit2x-0.16. The .dvi and .info documentation files are both there. Will Partain <partain@dcs.gla.ac.uk>*

From: **Dominique de Waleffe**
Date: 31 Aug 1993

Christian Lynbech writes: Texinfo should do the trick on straight text output. There is even the added feature of being able to produce an info file, i.e. a hypertext representation of the program! I have had this idea in the back of my head for quite some time, but I haven't had time to investigate it much.

I've had this in mind for a while and have been considering making nuweb work also for texinfo files. I think it would not be hard to quickly do if one abandons some of the nice formatting stuff that are available under Latex. But then it seems possible to exploit the referencing features of texinfo to provide navigational aids to browse the source under info.

*Certainly, producing a *usefull* program.info file, takes some care and consideration.*

In the project I'm working on, all system, manuals and program documentation must be written using TeXinfo. Then as usual that allows discrepancies (often quite large) between the real code and its associated documentation. I started using nuweb to provide additional documented code for the project, but now I would like to convert this to texinfo but without loosing the literate programming approach. This would allow me to convince other people of the benefits.

I am not much of a texinfo hacker myself, but since texinfo is a juiced down version of TeX (sort of), wouldn't that also solve some of the learning problems reported by people.

As I said earlier, texinfo is rather limited in typographical capabilities while the literate approach benefits a lot from such typography to include navigational references in the printed output while keeping those non-annoying (verbosity,length,size...)

Incidently, there is also a LaTeXinfo package, for [text deleted] The downside is that its maintainance situation is a [text deleted]

That's the reason why texinfo was chosen for the project mentioned above, but we did look at latexinfo too and found that it would allow better looking documentation to be produced. Maybe we could start a discussion on what would be required/nice/ideas related to modifying nuweb to support texinfo. (I already have a version that can use a character other than @ for commands)

From: **Dave Love**
Date: 09 Sep 1993

*Yes, I know about dvi2tty and such utilities. Theoutput fromthese isunacceptable, probably dueto kerningit lookslike this... Without a *lot* of mucking around with the text, it looks terrible. Nroff does a fair job of coming up with a straight text representation.*

If this sort of thing is important, I think you *can* do a reasonable job with (La)TeX, but you have to use an appropriate style and fonts to produce the dvi file, although I don't know of an entirely satisfactory solution at present. I may end up doing some work on this in the near future as a means of avoiding TeXinfo. If anyone really feels a need for it, let me know.

From: **Richard Kooijman**
Date: 09 Sep 1993

*If this sort of thing is important, I think you *can* do a reasonable job with (La)TeX, but you have to use an appropriate style and fonts to produce the dvi file, although I don't know of an entirely satisfactory solution at present. I may end up doing some work on this in the near future as a means of avoiding TeXinfo. If anyone really feels a need for it, let me know.*

I use dvidoc myself and I am very satisfied. As long as you need ASCII output, you put in a style called 'dvidoc', run LaTeX and use dvidoc to display the material. For final output you remove 'dvidoc' style and use your regular dvi2something.

---

comp.programming.literate passes 354:31

From: **Matthias Neeracher**
Date: 16 Sep 1993

I'm very happy about this result and from the vote result, I assume that I'm not the only one :-) Unless anybody posts a serious objection to news.groups within the next days, comp.programming.literate will be created in about 5 days. For bandwidth reasons, I omitted the full list of votes. Those interested may retrieve it by ftp from ftp.uu.net in the file usenet/news.announce.newgroups/comp/comp.programming.literate or by asking me for a copy by email. Hope to see you soon on USENET.

comp.programming.literate group vote results - 385 votes

Yes No : 2/3? >100? : Pass? : Group

---- ---- : ---- ----- : ----- : ----------

354 31 : Yes Yes : Yes : comp.programming.literate

CHARTER

Charter: A forum for the discussion of issues related to literate programming.

(1) To share ideas, questions, experiences, and knowledge about the reading and

writing of literate programs.

(2) To discuss the merits of the currently existing literate programming tools.

(3) To discuss the design of new literate programming tools.

If a newsgroup is created, it will be mirrored to the existing mailing list LitProg@shsu.edu. For reference purposes, the newsgroup will be fully archived by the host sponsoring the mailing list.

From: **George Greenwade**
Date: 22 Sep 1993

Unbelievable. In just the first few hours of its existence, comp.programming.literate (c.p.l) has been at least as active as LitProg when it first fired up and virtually all of the posts are from "newbies" (defined as not prior LitProg subscribers). As owner of LitProg, please allow me to intervene on a few topics which have already arisen. Chris Gray asked about the "Subscription to mailing list LitProg" post:

> I am a little confused by this posting --- are comp.programming.literate subscribers being invited to join the LitProg list, or is the newsgroup a mutation of the list, or what? Or maybe this LitProg document is intended to serve as a FAQ for comp.prog.lit? (Are abbreviations allowed on this group or are they "considered illiterate")?

LitProg was begun July 19, 1992, as the sole (as far as anyone involved in its initiation knew, anyway) source of discussion focused exclusively on literate programming. I don't believe that anyone involved expected that the list would grow as it did, nor that its subscribership or activity would get anywhere close to where it did as quickly as it did. To give credit where it's due, Cameron Smith

was the original proponent of the list, followed closely by Don Hosek. The core group (who are too numerous to list, but are very appreciated) created one of the better lists, IMO, on the nets.

Due to the growth of LitProg and a few side discussions on the list, Matthias Ulrich Neeracher posted the RFD and CFV for c.p.l, which is completely gatewayed with LitProg. Anything posted to LitProg@shsu.edu (LitProg@shsu.bitnet) will get to c.p.l (so those who wish to use news in lieu of mail may do so) and anything posted to c.p.l will get to LitProg (so those who wish to use mail instead of news or because they lack access to news may do so).

Next, the thread of "What does literate programming mean to you?" has already arisen (which I sort of expected). Please allow me to point out to news users (especially those with no prior contact with LitProg!) that this has been thrashed out previously on LitProg (not that it doesn't deserve rethrashing -- the *concept* of literate programing may be one of the more important items in gaining wider acceptance to literate programming as a real life production vehicle for programmers).

For reference, LitProg is fully archived (and I may soon begin a complete Gopher-based index-searchable archive -- this will require a little more time than I have at present, but I have the tools available to do so). You can retrieve the LitProg archives via ftp from Niord.SHSU.edu (192.92.15.8) in the directory [FILESERV.LITPROG] (Niord is a VMS machine). The files LITPROG.yyyy-mm are the archives, with "yyyy" being the year and "mm" the month (i.e., this post will be included in LITPROG.1993-09 since it's now September, 1993).

Next, Marcus Speh has already pointed out his World Wide Web literate programing library at info.desy.de (Marcus, please check your directory specifications for the files at SHSU, BTW -- I just tried and failed to get back here and my server logs indicate it is due to a bad request at your end). In addition to the WWW resource Marcus provides, Niord.SHSU.edu runs a Gopher server on port 70. I just moved a "Literate Programming Library" entry to the top-level menu there (it is still included in the "TeX-related Materials" menu for those of you who are used to getting to it from there). This gopher burrow at SHSU includes the archives of LitProg, access to the web/ directory on SHSU's host of the Comprehensive TeX Archive Network (CTAN), and access to most literate programming tools available from SHSU (and if you know of one or more I've omitted, please let me know and I'll get it in place). I just added an HTML link to Marcus' host so those of you with gopher clients which are HTML compatible can get there in one jump from Niord's Gopher server. I plan on working with Joachim Schrod (who you'll come to know as a valuable resource, if you don't already) on getting a parallel literate programming library arranged from and between his host in Darmstadt (Germany) and one of mine here (and make every effort to make it gopherable from here, as well). In other words, there is already a wealth of information and resources available for your use (and further development!).

Inline comments

From: **Stuart Ferguson**
Date: 18 Oct 1993

Stuart Ferguson wrote: A good example of how literate programming tools break down is the example CWEB text that Knuth himself provides. In his toy example of a word count program, the code scraps often contain ordinary C-style comments. These comments (not even well typeset) indicate the need for a finer granularity of abstraction than that provided by the scrap mechanism. In-line comments allow descriptions of fragments of scraps that CWEB does not.

C. M. Sperberg-McQueen replies: I am finding it hard to understand how your logic works here: as a demonstration that literate programming tools cannot handle comments on fine-grained details, you point out that Knuth's web for wc contains (how odd) comments on fine-grained details. Since CWEB allows inline comments, how is it possible for "In-line comments [to] allow descriptions ... that CWEB does not"?

It seemed like a step backwards to me to have a nicely typeset document describing a program and then have the code scraps contain untypeset comments in the code as if it were going to be read by a machine. Part of what I expect from a literate programming tool is the ability to write prose which describes each and every important detail of a program at the level at which it occurs. For some reason, Knuth chose to use inline comments to describe portions of his code instead of TeX, and I think he did this not because it was a good way to write commentary, but because the alternative was worse. He could have added another scrap for each item which he felt needed comment, but I would guess that Knuth thought that would break the code up too much. This suggests a problem to me -- that the scrap mechanism is not sufficient for properly explaining all the details in a C program.

Saying that "CWEB allows inline comments" is not the same as saying that CWEB handles inline comments *well*. It's a small point, and it wasn't the main thrust of what I was trying to say, but it really did bother me. Those few C-style comments seemed to poison the whole idea of programs written for people and not compilers.

From: **Bart Childs**
Date: 18 Oct 1993

It was written that the ordinary C-style comments were not even well typeset. I have no idea what was meant by that unless it was desired to have them not typeset at all. What was done is the comments are formatted like the documentation portions of sections.

There are some places where this style of comment is a great help and far better than having an excess of trivial sections. A good example is section 11 of TeX.web. DEK declared a large number of constants and a comment on each. That comment often includes a wonderfully informative statement about relationships that must exist between other constants... These comments make extensive use of the same `escape to TeX mode' that also exists throughout his webs. Most will reference variables.

This kind of precision is a great help. I remember being confused in my early reading about unix. The feature of being case sensitive is often mentioned and then when a command starts a sentence it is capitalized! I also note that he called these small parts sections or modules. (He used the words interchangeably.) I think that is a far better term than scraps because they were intentional, not leftovers as in most of the definitions of scrap.

From: **Joachim Schrod**
Date: 18 Oct 1993

Stuart Ferguson writes: It seemed like a step backwards to me to have a nicely typeset document describing a program and then have the code scraps contain untypeset comments in the code as if it were going to be read by a machine. [...] For some reason, Knuth chose to use inline comments to describe portions of his code instead of TeX,

Sorry, but I don't understand you. The inline comments of CWEB are of course in TeX mode, and therefore you can make use of TeX markup there as well. If it's good style (and good looking ;-) to put a picture environment there might be questionable, but it's possible. I prefer to regard inline comments as the footnotes of program code: Used with care they can enhance the understanding of the document at large. Btw, that's no new feature. WEB did it already. So, would you please elaborate your critique that CWEB/WEB/FWEB inline comments are (a) not TeX material, and (b) useless?

From: **Lee Wittenberg**
Date: 19 Oct 1993

I'm a bit confused. A number of people have made the claim that CWEB does not typeset comments. As far as I know, CWEB typesets comments exactly as it does the text sections: in roman type, with TeX codes completely usable. The cwebmac.tex macros also include options to typeset comments in sans serif and to use special symbols as comment markers. Am I missing something here?

From: **Stuart Ferguson**
Date: 19 Oct 1993

Stuart Ferguson writes: It seemed like a step backwards to me to have a nicely typeset document describing a program and then have the code scraps contain untypeset comments in the code as if it were going to be read by a machine. [...]

Joachim Schrod replies: Sorry, but I don't understand you. The inline comments of CWEB are of course in TeX mode, and therefore you can make use of TeX markup there as well. If it's good style (and good looking ;-) to put a

I could easily be mistaken on this point, as I only looked at the Knuth articles for a short time. I was convinced that the inline comments were printed in teletype font between "/*" and "*/" with a very distracting word wrap. I'll go back and see if I can find that article again. In any case, the point is somewhat irrelevant, since CWEB apparently *does* support typeset inline comments. I'm pleased to hear this, since I think this is an important feature for literate C programing.

What does literate programming mean to you?

From: **Craig Hubley**
Date: 21 Sep 1993

Of course I am familiar with the original idea but to me it seems like an evocative term so I wonder what other definitions it suggests. To me, "literate programming" implies that all information relevant to the construction of the system is embedded within it. Ideally this would work in a hypertext fashion, so that I can easily trace requirements, find descriptions of limitations where they are imposed (i.e. in design, or in a particular implementation). It makes it simple to answer questions posed at any point in the code like "why is this here?" or "what is this doing?" or "what else is affected by this?". In other words, perhaps to me the most important thing about literate programming is the glossary and the index... :)

From: **Freeland Abbott**
Date: 21 Sep 1993

Craig Hubley writes: In other words, perhaps to me the most important thing about literate programming is the glossary and the

index... :)

You forgot the guarantee. "What guarantee?" you ask... I mean the guarantee that the information is up-to-date and therefore likely to be correct; it's the difficultly of providing that which makes the job hard.

From: **Eugene Miya**
Date: 21 Sep 1993

I just visited the newly moved Comp. Lit Bookshop now at the Apple R&D Center. I picked up DEK's book based on a discussion we had over dinner a couple months ago. Literate Programming means (to me) Don's book on the subject. It's a "Fabrege egg" to quote Doug McIlroy in my Favorite Programming Pearl with Don, Doug, and Jon Bentley. It's an unfinished Art of Computer Programming with a highly acclaimed text processing system which makes documents look like assembly language. It's The Errors of TeX. It lacks the empirical study of Fortran programs. It avoids the hype of topics like AI and virtual reality. It requires an informed, well-read readership.

From: **Tim Larkin**
Date: 22 Sep 1993

Of course I am familiar with the original idea but to me it seems like an evocative term so I wonder what other definitions it suggests.

To me, literate programming means writing a program text for a human reader rather than for a compiler. Hypertext, index, glossary, pretty printing, these are tactics, not to be confused with the goal. A literate programmer's first priority isn't to be clever or efficient: his first priority is to explain the problem and the solution so that a human reader will understand them and will enjoy learning about them. As Horace counselled, to entertain and to instruct.

From: **Huaiyu Zhu**
Date: 22 Sep 1993

Freeland Abbott writes: You forgot the guarantee. "What guarantee?" you ask... I mean the guarantee that the information is up-to-date and therefore likely to be correct; it's the difficultly of providing that which makes the job hard.

The difficulty arises from the fact that computers and humans read completely different parts of a literate program. Following are some ideas of how this can be avoided. At present we have 'pure programing languages' like C, 'pure text formatting languages' like TeX. Their combination is the so-called literate programs. However, we also have many computer algebra software, (also called symbolic computation). If they are added to the programs, the results can be far more robust against change. Here's a sample C function.

```
---------------
void f(float a)
{
                    //requires: a>0
    float b, c, d;
    b = sqrt(a);    //guaranttees: b>0
    scanf("%f", c); //requires: c>0
    d = c+1/c;      //guaranttees: d>2
                    //invariant: a
}
--------------
This is what an "ideal compiler" should say:

assertion at line 3 (a>0) passed to linker.
assertion at line 6 (c>0) changed to run time checking.
assertion at line 7 (d>2) not guarantteed. Best possible: d>=2.
_________________
```

In this way, the compiler reads the program and assertions, and make sure that they agree with each other. The typesetting mechinism makes the assertions part of the *text*, so that humans can read them easily. Of cause this idea comes from the programming language Eiffel, but what I would like is a full symbolic computation mechanism. It will be really great if there is a universal mechanism by which programing language, symbolic computation software, and text formatting software can be combined, even if they are designed without regard to each other. Sounds like a dream?

From: **Michael Koopman**
Date: 22 Sep 1993

Tim Larkin wrote to LitProg (a.k.a. comp.programming.literate): To me, literate programming means writing a program text for a human reader rather than for a compiler.

Some of the webs written for literate programming tools do not appear to be enhanced expositions of the principles underlying the behavior of the program (while including the code scraps). Such leading, literate program authors could be considered hypocrit infidels not practicing what they seem to preach (false names are used to protect the innocent). A comprehensive and well structured presentation (printed document, hypertext, other) that compiles to realize the description is what I expect from literate programming.

Techniques that may benefit web reading and writing and have not been discussed (my wish list) include: An outline mode - with multiple levels of exposition. A hypertext tree allowing for "outlines" for different points of view (audiences). Chart, figure and graphics support capabilities (perhaps, SGML). { OK, TeX does this - but it is 'non-trivial' with TeX or LaTeX] Also, audio and other presentation media inclusion.

The web tools I have seen allow the literate programmer to improve (or demote) modularity, localization and other principles of software design. The structure of the web (hopefully, with "meta-comments") depicts the implementation of these design principles in its totality. Maintainability (and reusability) can also be improved (or reduced) with a web design. These latter principle seems to be the ones which literate programming can improve most. Providing short, relatively cryptic titles for code scraps and providing no discussion on the algorithm and implementation which the code realizes is not only opposed to what I expect from literate programming but is not literate programming, at all, IMNSHO.

From: **Freeland Abbott**
Date: 22 Sep 1993

Huaiyu Zhu writes: The difficulty arises from the fact that computers and humans read completely different parts of a literate program. Following are some ideas of how this can be avoided.

```
---------------
void f(float a)
{
                    //requires: a>0
    float b, c, d;
    b = sqrt(a);    //guaranttees: b>0
    scanf("%f", c); //requires: c>0
    d = c+1/c;      //guaranttees: d>2
                    //invariant: a
}

--------------
This is what an "ideal compiler" should say:

assertion at line 3 (a>0) passed to linker.
assertion at line 6 (c>0) changed to run time checking.
assertion at line 7 (d>2) not guarantteed. Best possible: d>=2.
_______________
```

Well, first of all, *my* ideal compiler would complain about the syntax error on line 6, where you want &c. ;-) Um... although I agree that this is part of the difficulty, I don't think it's all of it, and perhaps not even the hardest part... although your hypothetical compiler's figuring out not only that line 7 isn't guaranteed, but also that d >= 2 is the best approximation, is mighty impressive. But the difficulty also comes from the fact that computers have difficulty grasping high-order concepts, and *that* is what people are usually interested in. So, although your example is fine as far as it goes, I'd rather be able to make assertions like "the queue contains no unmarked events" or "none of these objects are mapped to the screen" or the like. I can look at your line 7 and see (with only a very little thought) that yes, d must be at least 2.0. It's harder to look at an arbitrary function call (with arbitrary subcalls) and know what that will guarantee, in semantic as well as mathematical terms.

Having programmatic checking of the mathematical effects is certainly computable, of course---if, in your example, d were a global variable, for example, a program could certainly know that a call to f() would guarantee that the global d was at least 2.0 after the call. But imagine that the number were instead added to some sort of (dynamically allocated, globally scoped) list which recorded c+1/c for a number of inputs c, which would then be manipulated. Various mathematic constraints could be proven, certainly, but they would tend not to capture the notion "d, which is at least 2.0, is in the list after a call to f()".... it'd be more like "the value field of the struct node pointed to by listHead is at least 2.0, and the next field of the struct is either null or non-null; if it is non-null, the value field of the struct node pointed to by listHead is at least 2.0, and the next fiend of the struct is either...," which would continue infinitely.

So, how do you catch the semantics of a singly-linked list in a scheme like this? The constraint I'd like to get out of that version of f() is "after a call to f(), the list will not be empty, the first element of it will be the sum of most recent input and its multiplicative inverse, and the rest of it will be the previous list."

Sounds like a dream?

Depends: the not-very-useful mathematical form doesn't, although it sounds hard (especially relative to its benefit to me). The more useful semantic form, yes, sounds dream-like.

From: **Bradley Sherman**
Date: 22 Sep 1993

The literate programmer writes code that I, another literate programmer, can understand. Polished in her style, she can anticipate the areas of the code most likely to be altered and provides comments in tight grammatical English. The literate programmer knows that the code is the essence of the program. The commentary, code and whitespace are in harmony. He is obsessed with making good lexical choices. The literate programmer knows that copyright protection of source code is absurd; knowing the right approach to the problem --and which problem to approach-- is everything. The literate programmer is a poet not a novelist.

From: **Stuart Ferguson**
Date: 23 Sep 1993

Freeland Abbott writes: You forgot the guarantee. "What guarantee?" you ask... I mean the guarantee that the information is up-to-date and therefore likely to be correct; it's the difficultly of providing that which makes the job hard.

I say phooey. "What guarantee?" indeed. What guarantee do I have for any given random piece of code that it operates correctly? I have the word only of the original programmer, and their reputation which I must evaluate and trust or not. In a world of literate programmers, keeping the information up to date and accurate is part of the job. Innacurate commentary is unprofessional. Dammit.

From: **James Foster**
Date: 23 Sep 1993

Stuart Ferguson writes: I say phooey. "What guarantee?" indeed. What guarantee do I have for any given random piece of code that it operates correctly? I have the word only of the original programmer, and their reputation which I must evaluate and trust or not. In a world of literate programmers, keeping the information up to date and accurate is part of the job. Innacurate commentary is unprofessional. Dammit.

This led me to an interesting thought (quick, bring an ice pack!)... Why not include a formal proof that the code has been verified, along with an informal description of what it does (you know, traditional unprofessional documentation)? A good untangler could strip the proof, if it exists, and run it through a theorem verifier in order to certify it. Now you need only check the English documentation against the assumptions in the verification proof. Before everyone gives the usual complaints against formal verification, let me remind you that 1) they don't matter, we're discussing literate programming and a proof should be part of this if it exists; 2) most of those arguments are obsolete; 3) sometimes there is no alternative to formally verified code (ever tried to debug a pacemaker?).

From: **Marcus Speh**
Date: 23 Sep 1993

The literate programmer writes code that I, another literate programmer, can understand. Polished in her style, she can anticipate the areas of the code most likely to be altered and provides comments in tight grammatical English. The literate programmer knows that the code is the essence of the program. The commentary, code and whitespace are in harmony. He is obsessed with making good lexical choices. The literate programmer is a poet not a novelist.

Very nice account on literate programming, I like that!

The literate programmer knows that copyright protection of source code is absurd; knowing the right approach to the problem --and which problem to approach-- is everything.

Agreed, cum grano salis: I do put a copyright under the GNU "copyleft" which I think is suitable also for literate programs. This copyleft is transferred to Global Network Academy, Inc. (Usenet University), an educational institution, thus not violating "Bradley's Law" :-)

From: **Preston Briggs**
Date: 23 Sep 1993

Michael Koopman writes: Some of the webs written for literate programming tools do not appear to be enhanced expositions of the principles underlying the behavior of the program (while including the code scraps). Such leading, literate program authors could be considered hypocrit infidels not practicing what they seem to preach.

Hmm. Certainly nuweb (I wrote it) is worse in this respect than any other tools I've examined. So why did I write such an illiterate program? I have to consider it incomplete. It'll get more complete as I spend more effort on it. In the meantime, I (and others) are able to use it without worrying about whether I've finished describing it all. It's just a tool, not an end in itself. A hammer doesn't need a dissertation attached to be useful.

From: **Norman Ramsey**
Date: 23 Sep 1993

Some of the webs written for literate programming tools do not appear to be enhanced expositions of the principles underlying the behavior of the program (while including the code scraps). Such leading, literate program authors could be considered hypocrit infidels not practicing what they seem to preach (false names are used to protect the innocent).

I confess that my tool, noweb, is at best of poor quality when considered as a literate program. In mitigation I offer these two observations: i) Making a good-quality literate program is 3-4 times as expensive as just making a working program using literate-programming tools. ii) I have been unable to develop really good literate programs without peer review. I tolerate my ugly "literate" programs because in my work a program is seldom an artifact of the first importance. Far more often the paper is what matters (and what gets polished).

From: **Michael Koopman**
Date: 23 Sep 1993

Preston Briggs: So why did I write such an illiterate program? Norman Ramsey: I confess that my tool, noweb, is ...

As a novice and a "father confessor," concurrently :-) .... Both of your works are certainly valuable literate programming tools. Perhaps your self-evaluations are a bit critical with regards to these webs as examples. The structure of the web (irrespective of the depth of exposition in English [other]) and the code scraps provide the reader versed in the programming language quite a bit to go on. Simply the structure of the weave (index, cross ref.s) is a blessing compared to "illiterate" programs.

Certainly, a working tool with a minimal description is more valuable than a description sans working implementation. Are these tools examples which prove that literate programming is only a benefit in certain cases? How is the crossover point to be determined? Expected life and audience of the program should be factors.

Norman Ramsey: i) Making a good-quality literate program is 3-4times as expensive as just making a working program using literate-programming tools. ii) I have been unable to develop really good literate programs without peer review.

How does a literate programmer address these isues without risking intellectual property and justifying costs?

From: **Preston Briggs**
Date: 23 Sep 1993

Are these tools [nuweb and noweb] examples which prove that literate programming is only a benefit in certain cases? How is the crossover point to be determined? Expected life and audience of the program should be factors.

I think nuweb is better expressed in nuweb than in raw C. And I think it'll be even better in the future. However, your question remains: Are there programs that would not benefit? For myself, I'm not sure. I generally use nuweb now instead of plain C, even when prototyping. On the other hand, I don't use it for Makefiles (though I might if I ever did anything hard).

I think your point about "audience" is very important. Certainly we're constantly told to consider the intended audience when writing ordinary papers. And we obviously consider the audience when speaking (e.g., how you speak to your parents, your colleages, your children, a waiter, etc). However, I'm sure what advice to give. I feel very much like I'm learning as I go.

From: **Eric Johansson**
Date: 23 Sep 1993

Stuart Ferguson writes: I say phooey. "What guarantee?" indeed. What guarantee do I have for any given random piece of code that it operates correctly? I have the word only of the original programmer, and their reputation which I must evaluate and trust or not. In a world of literate programmers, keeping the information up to date and accurate is part of the job. Innacurate commentary is unprofessional. Dammit.

James Foster writes: This led me to an interesting thought (quick, bring an ice pack!)... Why not include a formal proof that the code has been verified, along with an informal description of what it does (you know, traditional unprofessional documentation)? A good untangler could strip the proof, if it exists, and run it through a theorem verifier in order to certify it. Now you need only check the English documentation against the assumptions in the verification proof. Before everyone gives the usual complaints against formal verification, let me remind you that 1) they don't matter, we're discussing literate programming and a proof should be part of this if it exists; 2) most of those arguments are obsolete; 3) sometimes there is no alternative to formally verified code (ever tried to debug a pacemaker?).

Another variant on this theme is inclusion of a formal derivation of a program. for examples, look at the Z specification methodologies (set based) and Ed Cohen's book, "Programming in the 1990's" (predicate based) I am a fan of formal derivation of programs because it is something we can do today instead of the someday promised by the verification folks.

In the informal "what can we do now" vein, I think literate programming can help trace requirements from the initial requirements document to the actual implementation because indexing/cross reference capability present on the text processing side of the house. trying to track what code satisfies what requirement is a laborious manual process that mostly counts on a coder's memory of the code and the requirements document.

From: **Eitan Gurari**
Date: 24 Sep 1993

Literate programming offers me a natural medium for 1. Communicating information between different phases of code development. 2. Arguing with myself about my code by explaining its meaning. Since an explanation of subject matter is a description of an understanding of the subject matter, a programmer who provides an explanation for code must closely inspect his or her understanding of the code as the description of this understanding of the code is being produced. Such a mode of operation encourages programmers to take a critical look at their code, and it results in code that is prepared with a lot of care.

Macro-based abstractions with natural language oriented titles for code segments, fragmentation of code segments, prose, figures, and mechanisms to structure documents are the ingredients that are important to me in the medium that I am using.

I consider it a worthless effort to polish the exposition and appearance of literate programs that are not intended to be consumed by others (i.e., documents that are not intended for peer review---see norman's note). Consequently, documents that serve as literate programs to me might be cryptic creatures to other readers.

From: **Zen**
Date: 24 Sep 1993

Michael Koopman writes: Some of the webs written for literate programming tools do not appear to be enhanced expositions of the principles underlying the behavior of the program (while including the code scraps). Such leading, literate program authors could be considered hypocrit infidels not practicing what they seem to preach.

Preston Briggs writes: Hmm. Certainly nuweb (I wrote it) is worse in this respect than any other tools I've examined. So why did I write such an illiterate program? I have to consider it incomplete. It'll get more complete as I spend more effort on it. In the meantime, I (and others) are able to use it without worrying about whether I've finished describing it all. It's just a tool, not an end in itself. A hammer doesn't need a dissertation attached to be useful.

Talk about justifying your actions... you could say the same thing about any tool or program -- they're just not finished. I've had to use, modify, and throw away countless programs that aren't commented or are poorly written, just because the author(s) thought that it wasn't worth the effort or that they'd get around to it later. One of the worse things you can do to a program is to comment/document it (or "literalize" it?) *after* it's a working piece of code. IMHO, writing literate or even just a "good" program (if you can consider a program to be good if it doesn't follow the literate standards set by knuth and others) very nearly *requires* you to have as good design as possible from the start, and then to follow the same standards in every phase of the code that you want to have when it's a finished product at the end; tacking it on at the end is something that is almost never gotten around to, and is almost guaranteed to be of lesser quality than something that was done right the first time.

I (and others) are able to use it without worrying about whether I've finished describing it all

Why ever bother making it literate, then, if it works fine without it?

Bradley K. Sherman writes: The literate programmer...

I would hope, in addition to everything else, the literate programming would be concerned that programs written in the literate style actually work (but see below, I don't know).

The literate programmer knows that copyright protection of source code is absurd; knowing the right approach to the problem --and which problem to approach-- is everything.

Copyrights are absurd? Do you mean algorithm copyrighting or that *all* code should be free to everyone else? What does the latter have to do with literate programming?

The literate programmer is a poet not a novelist.

Poetry can obscure meanings. I'd rather read code written in the style of dickens rather than e.e. cummings... to me, I think the key is functionality and clarity first, not style. I wonder if people here view literate programming as more of an art form, something that has inherent usefulness, or as a vehicle to produce "better" (whatever that means) programs.

Norman Ramsey writes: ii) I have been unable to develop really good literate programs without peer review.

Why is that? Is it too hard to recognize literacy when seen, or is it that you don't have enough practice, the tools aren't there, or ...?

From: **Chris Gray**
Date: 24 Sep 1993

Michael G. Koopman writes: Some of the webs written for literate programming tools do not appear to be enhanced expositions of the principles underlying the behavior of the program (while including the code scraps). Such leading, literate program authors could be considered hypocrit infidels not practicing what they seem to preach (false names are used to protect the innocent). A comprehensive and well structured presentation (printed document, hypertext, other) that compiles to realize the description is what I expect from literate programming.

To which norman@bellcore.com, author of noweb, retorts: I tolerate my ugly "literate" programs because in my work a program is seldom an artifact of the first importance. Far more often the paper is what matters (and what gets polished).

Well that's a pretty straightforward admission that to this guy at least literate programming is just an academic fad which enables you to get papers published, not a serious proposal to enhance the quality of our programs (and our lives). If that's true then I'm wasting my time reading this stuff and I should get back to work mungeing my design documentation into comments in the source code. Apparently Preston Briggs also feels his ears burning:

Hmm. Certainly nuweb (I wrote it) is worse in this respect than any other tools I've examined. So why did I write such an illiterate program? I have to consider it incomplete. It'll get more complete as I spend more effort on it. In the meantime, I (and others) are able to use it without worrying about whether I've finished describing it all. It's just a tool, not an end in itself. A hammer doesn't need a dissertation attached to be useful.

To which I say: phooey (tm). Games and demo's aside, all programs are tools. Lotus 1-2-3 is a tool, Cubase and KCS are tools. The programs which programmers call "tools" are the ones which they use to make other programs. It is precisely these programs which people are most likely to feel the need to hack into and rearrange and modify in order to provide support for their company documentation scheme, network file server, GUI, and Nintendo Power Glove. These are programs written by programmers for programmers, and they _should_ be exemplary.

From: **Osman Buyukisik**
Date: 24 Sep 1993

*zen@death.corp.sun.com said: Talk about justifying your actions... you could say the same thing about any tool or program -- they're just not finished. I've had to use, modify, and throw away countless programs that aren't commented or are poorly written, just because the author(s) thought that it wasn't worth the effort or that they'd get around to it later. One of the worse things you can do to a program is to comment/document it (or "literalize" it?) *after* it's a working piece of code. IMHO, writing literate or even just a "good" program (if you can consider a program to be good if it doesn't follow the literate standards set by knuth and others) very nearly *requires* you to have as good design as possible from the start, and then to follow the same standards in every phase of the code that you want to have when it's a finished product at the end; tacking it on at the end is something that is almost never gotten around to, and is almost guaranteed to be of lesser quality than something that was done right the first time.*

I can think of one reason to literalize it: so that it may the first example for a new user, and also you tend to find bugs or better ways of doing the same thing, or so that you can, at a later time, understand what you did without spending a lot of time. Even though Preston(nuweb's author) thought nuweb was not literate, I had no problem with it. It was an example of how to use the tool, and I was able to understand what he did even though I am not a programmer (engineer)!

From: **Preston Briggs**
Date: 24 Sep 1993

zen@death.corp.sun.com (somebody?) writes: Talk about justifying your actions... you could say the same thing about any tool or program -- they're just not finished.

Of course I justify my actions; what did you expect? And nuweb does keep improving, both in functionality and documentation. When it's finished, I won't work on it any more.

*One of the worse things you can do to a program is to comment/document it (or "literalize" it?) *after* it's a working piece of code.*

Why? It may not turn out as well as a beutiful example, worked up from scratch, but it won't hurt the code. In the case of nuweb, I wrote the spec first, then an initial cut at the code in CWEB. Once it would handle itself, I translated the web into nuweb and continued from there (why? as an experiment to see what it was like to use nuweb.) At some point, I said to myself "This is cool, I like it, I wonder if others will?" and I started giving it away. This garnered a lot of comments and ideas as more people used it (and modified it) and I've been able to build some of the better ideas into the system. Eventually it'll settle down, as it either approaches perfection or bumps up against the basic limitations of its approach. When one these things happens, I'll either rewrite it, rationalizing some of the code, or I'll chunk it and begin using a better tool.

Why ever bother making it literate, then, if it works fine without it?

I wrote it in nuweb (and CWEB initially) so I could practice using web, so I could experiment with different ways of explaining things, so I (and readers) could benefit from the indices and crossreferences and free code arrangment, and so I could take advantage of the features of Latex to help explain my code.

From: **Zen**
Date: 24 Sep 1993

Preston Briggs writes: Of course I justify my actions; what did you expect?

No less, of course. The point I was trying to make was that your justification was being hypocritical to the (*grin* -- well, *my* ideal, of course!) ideal of literate programming.

*One of the worse things you can do to a program is to comment/document it (or "literalize" it?) *after* it's a working piece of code. Why? It may not turn out as well as a beutiful example, worked up from scratch, but it won't hurt the code.*

It almost certainly will. IMO, literalizing the code is an integral part of the design and coding process, something that grows as the program does. It helps you write a better program; that's the point. When you try to go back post facto and slap it in, you have several problems (off the top o' my head): 1) It probably won't get done. Very rarely are working programs rewritten -- time is too much of a factor. 2) If, as is often the case with most programs, it doesn't do what you want, you have to start again from scratch, then you never get around to writing a good program. 3) You lose the advantage of doing it right from the start; if you accept the premise that the program will be better, both from a design and coding standpoint if literalized, then you lose both the extra time you take to do the job twice *and* the advantages of design that you gain from the literalizing. 4) WRT quality, it will be all the worse for not being literal (until you run back and make it right.) People won't use it or accept it as much, and the original programmer(s) won't have incentive to improve it. 5) If no one writes literate programs, then no one will see the advantages. You throw something together, it works, and

everyone is happy, right? Well, "works" is a very subjective thing. If I write a program I don't usually want it just to work for me. I want others to be able to use it, modify it, and understand it. To me, that's all part of the joy and purpose of programming. 6) It's early in the morning, but there are lots of other reasons...

In the case of nuweb, I wrote the spec first, then an initial cut at the code in CWEB. Once it would handle itself, I translated the web into nuweb and continued from there (why? as an experiment to see what it was like to use nuweb.) At some point, I said to myself "This is cool, I like it, I wonder if others will?" and I started giving it away. This garnered a lot of comments and ideas as more people used it (and modified it) and I've been able to build some of the better ideas into the system.

You would have gotten more and better comments if people could have *read* what you wrote and how you implemented your decisions. Someone said that they couldn't write a literate program without peer review; I claim that in most nearly all cases that you're not going to get that peer review if you don't write it literally.

Eventually it'll settle down, as it either approaches perfection or bumps up against the basic limitations of its approach. When one these things happens, I'll either rewrite it, rationalizing some of the code, or I'll chunk it and begin using a better tool.

sigh That's what happens to code. People write it, use it, then move on. Nothing ever gets done *right*, it's all just another justification for not literalizing it. If you had done this right, I maintain that not only would the chances of you scrapping it and starting from fresh be greatly decreased, you would learn a lot more on how to design and implement this sort of program. A quick reference -- have you read knuth and his experiences, when he first starts to program a large project, and why he believes in literate programming? Going back to your other article:

It's just a tool, not an end in itself. A hammer doesn't need a dissertation attached to be useful.

It's just an amazing statement, esp. on this newsgroup. But to address it again, your program isn't a hammer. It has lots of features, lots of ideas packed into it, lots of ways to use it by lots of people using lots of systems. But even that lowly hammer needs to be made right; constructing a good hammer means you have to consider what it's going to be used for, to choose the right materials. It means you have to communicate to the user what it can and should be used for (claw vs sledge hammers, for instance).

I find it amazingly ironic that a program to do literate program isn't written in a literate manner, but far more ironic that the programmer behind it thinks that it was the right thing to do.

From: **Bart Childs**
Date: 24 Sep 1993

Some comments by Bart Childs on recent postings. I agree with the characteristics that Bradley Sherman gave for literate programming. I have written several introductory documents on this that we use for introductory purposes. They are available for anonymous ftp from: ftp.cs.tamu.edu. /pub/tex-web/web/DOCs is the directory. I will email them to individuals who cannot ftp. They are too long for posting here.

Bradley noted that literate programs are written for other readers. That is easily overlooked (Norman Ramsey was eloquent about peer review in the LitProg discussion list.) Review will not automatically convert programs into publications (as in tenure and promotion documents.) He used the phrase "tight grammatical English" and there have been a number of suggestions that may be over simplified by just "hypertextize the specifications and the other 2167A documents." Samuel Johnson wrote: "What is written without effort is in general read without pleasure." The point is that quality documentation is never free or accidental. It takes work, work, ... Hypertext links to specs... can be valuable but is no replacement for real documentation of the thoughts behind programming decisions.

Foster@jed.cs.uidaho.edu stated "Why not include a formal proof that the code has been verified, along with an informal description of what it does (you know, traditional ... Before everyone gives the usual complaints against formal verification, let me remind you that 1) they don't matter, we're discussing literate programming and a proof should be part of this if it exists; ..."

Of course, if the proof exists it should be a part of the documentation or as a minimum a "reference" to its availability. Another item of the same type is that graphics and EVERY other aid to understanding how, why, ... should be part of a literate program. All that takes is work, work, ... (editing is work). What portion of our codes could be formally proven...? I guess that I am a cynic on that and would guess ... well, a small percentage.

Norman Ramsey and Preston Briggs have been modest about their work. Both have made statements about `simple tools.' Edsger Dijkstra repeatedly points out "if you want to make a user interface more difficult to use, add functionality to it." Ramsey and Briggs have created simpler tools (than DEK's original ...) I think the addition of the capabilities of multiple output files is a typical example of added functionality that is so easily handled otherwise. Sure, there is value in having a script that is associated with a huge code in the

same file, but if you change (say) the documentation of the script that should not imply the code needs recompilation too (in a make sense).

There have been several postings over the past year that indicate a desire for perturbations to major changes in literate programming systems. I think we don't know for sure what they should be like and we need a lot of documented use of the existing systems. I have not found that WEB (original, F, or C) is difficult for (even) students to learn. It takes a little training. We can't be sure of the contributions of literate programming until we write lots of programs, document their maintenance, and make them available for study. Norman Ramsey and Carla Marceau's paper (see the LitProg archives) is still one of the few papers about using WEB in a professional environment, several programmers ...

Stephen Savitzky stated: "My approach to literate programming is to write programs that can be read, rather than the Web approach of writing programs that can be processed to make them readable. I do this because I rarely, if ever, get a listing; I much prefer to edit on the screen. I want my programs to be as readable as possible when I do."

I am not saying that his codes are not wonderful expositions, but this indicates the goal is not the future reader, but his instant self gratification (apologies, if that is taken as a flame.)

Incidentally, there has been some previous discussion about the index or lack thereof in some literate programming systems. (Indexes of variable names obviously require knowledge of the HLL.) Some users have stated the use of the `formalized procedures of pseudo-code' (my words) as the greatest value of the whole literate programming process. I can understand this opinion at writing time, but years later at maintenance it will be greatly diminished. The use of the `formalized procedures of pseudo-code' is only a part of literate programming.

Aaron stated the status quo: "What difference does it make? If you think in code, write in code and generate documentation. If you think in documentation, write in doc and generate the code. If you don't think, don't write. :)"

The commonly stated statistic varies from 60% to 80% for the cost of the maintenance portion of a code in its lifetime. I think that literate programming can aid that.

Robert McLay started a discussion with this subject: Subject: Big Programs & Separate files & Make ... He offered some numbers that I wish to correct. TeX and METAFONT are each about 25k lines of Pascal code. (Count the semicolons and realize that most Pascal code has a significant number of lines without them too.) Both are written in the monolithic form required by the Pascal compilers of the early 1980's.

I recommend that literate programming be used in a manner that does not cause a dramatic change to the usual software life cycle for many developers. The only part that should really change is the code development part. Most developers use (something like) make and the dependencies just get one more item. Sure it slows it down a bit, but the paybacks are large in comparison (IMHO). Knuth stated it was true but never gave any statistics to prove it. He also keeps a detailed diary and could well have the data to prove it. Also see comment about Ramsey and Marceau above.

Mike Yoder described a tool (process?) that saves significant recompiling ... Each of these should always cause a reflection on the Dijkstra quote. Is the additional complexity of another tool ... worth it? How much is the cost of the additional (wasted) compiles?

Norman Ramsey writes: ii) I have been unable to develop really good literate programs without peer review. zen@death.corp.sun.com writes: Why is that? Is it too hard to recognize literacy when seen, or is it that you don't have enough practice, the tools aren't there, or ...?

I think the answer is two-fold. First, we are terrible judges of our own work, whether writing, dancing, speaking, programming, ... Secondly, the process of programming is intensive and often characterized by our having tunnel vision, not being able to find a forest because we keep bumping into trees, ...

I am sure there are people who can write well and don't need as much peer review. Much of Don Knuth's work would likely qualify. Careful reading of his work also shows that he gets it reviewed by as many people as he can. Most readers of this list will likely claim to be in environments where it is not practical to get said review.

Zen also discusses this in a later posting and expresses the desire to know more about why certain design decisions were made. These are available for Knuth's original WEB. Some of these are couched in the large variability of Pascal compilers at the time. This same info is available for CWEB and FWEB, but not all questions will be answered. His point is made.

Michael Koopman writes: Some of the webs written for literate programming tools do not appear ... To which norman@bellcore.com, author of noweb, retorts: I tolerate my ugly "literate" programs because in my work a program is seldom an artifact of the first importance. Far more often the paper is what matters (and what gets polished). Chris Gray writes: Well that's a pretty straightforward admission that to this guy at least literate programming is just an academic fad which enables you to get papers published, not a serious proposal to enhance the quality of our

M. Gray is 180 degrees out of phase. Norman has at least one paper (mentioned earlier) in literate programming, but his dissertation was on "Retargettable Debuggers" and his professional work is more in that line, distributed systems ..., I think. If literate programming is an academic fad, I would like to know the schools participating in this hype. I am sure that I would be one of the most guilty. I have taught a graduate `special topics' course on the subject twice and we are now experimenting with a freshman class. Although there are several journals that encourage papers on literate programming, few have appeared. Some of us are trying to change those small numbers, but fad it is not.

From: **Preston Briggs**
Date: 24 Sep 1993

zen@death.corp.sun.com writes a lot of stuff. I won't quote it. He says I'm a hypocrite for writing nuweb in a less-than-perfectly-literate fashion. I disagree. I'm not preaching literate programming, I'm just a user. I wrote nuweb and I use it. Since I also boss some programmers, I make them use nuweb. Some of my friends here use it, and many people in the newsgroup use it.

Why? Because it's got a nice combination of features. It provides value at very little cost. The question of how it's coded is an entirely orthogonal issue. In fact, I'd say there are 3 orthogonal issues here (and these apply to nuweb, noweb, CWEB, ...) 1) The "language" definition -- the features (and misfeatures) of the system. 2) The actual collection of tools that support the language (in the case of web, the tangle and weave programs, and maybe the TeX processor and Pascal compiler). 3) How each of the tools is implemented.

The interesting part to me, and perhaps others, is behind door number 1. Door #2 is also interesting to many people who'd like to use nuweb (or noweb or CWEB or whatever). They actually want to run the code! As a user, I have an interest in this part too. Door #3 is, in the case of nuweb, quite dull. It's a straightforward programming task and I wouldn't hesitate to give it to any undergraduate.

When I give out copies of nuweb and people send me comments, they send me, by and large, suggestions and complaints about the language and capabilities. This is great; just the sort of feedback I'm interested in. Sometimes people point out portability problems, usually with fixes. That's great too; we fix them and get back to the interesting questions. Nobody has commented on how I implement my state machines or how I parse command-line arguments or how I allocate memory. Why? Because those're just boring details. I'm not trying to teach people how to program, I'm trying to learn what makes a nice language for literate programming.

have you read knuth and his experiences, when he first starts to program a large project, and why he believes in literate programming?

Yes, I believe I've read everything Knuth has published about programming. I also believe I've read everything that's been published about literate programming. Why, praytell? Are we not to think and experiment for ourselves?

I also wrote: It's just a tool, not an end in itself. A hammer doesn't need a dissertation attached to be useful. zen@death.corp.sun.com notes: It's just an amazing statement, esp. on this newsgroup. But to address it again, your program isn't a hammer.

No, it's really an experiment with the idea of hammering, kind of like a big, flat rock. It's Preston, the caveman programmer, saying "Guys, look at this! You can beat on things even harder than with your fist. Hmmm... Do you think it needs a handle? Yeah, look! It's even better."

you have to communicate to the user what it can and should be used for (claw vs sledge hammers, for instance).

But nobody knows these things, least of all me. We're all very interested in what it might be good for and we find out more all the time. And someday, someone is going to have a drastically better idea and I'm going to let nuweb die. For example, noweb has a some ideas I really like. If, after some experience, I'm persuaded that his approach is better, I'll switch. It isn't a matter of how nuweb is written, it's a matter of the language design. In fact, I could fairly easily adapt my program to handle noweb source rather than nuweb source, so I'd claim (again!) that the tool and the language are seperate entities.

I find it amazingly ironic that a program to do literate program isn't written in a literate manner, but far more ironic that the programmer behind it thinks that it was the right thing to do.

If I wrote an assembler, I'd write it in a higher-level language. I am writing a Fortran compiler in C. Further, I don't write in either assembler or Fortran. The irony is just overwhelming!?

From: **Michael Koopman**
Date: 24 Sep 1993

zen@DEATH.CORP.SUN.COM responding to Bradley K. Sherman: The literate programmer is a poet not a novelist. Poetry can obscure meanings. I'd rather read code written in the style of dickens rather than e.e. cummings... to me, I think the key is functionality and clarity first, not style. I wonder if people here view literate programming as more of an art form, something that has inherent usefulness, or as a vehicle to produce "better" (whatever that means) programs.

Which is e.e. cummings and which is Dickens: 'C'syntax or CWEB? Certainly code scraps come on little cat paws and fog the purpose of the particular implementation. This is especially true of the seldom commented nuances; the "real" work the programmer labored over.

I would hope, in addition to everything else, the literate programming would be concerned that programs written in the literate style actually work.

Interpolating Preston Briggs: "It'll get more complete as I spend more effort on it. In the meantime, I (and others) are able to use it without worrying about whether I've finished describing it all. It's just a tool, not an end in itself. A hammer doesn't need a dissertation attached to be useful."

I view a program as an art form analogous to a symphony. There *is* elegant code. Preston Briggs discussed one of his "unfinished" symphonies. The symphony is played for the end user. Literate programming is just an improvement in the sheet music.

From: **Lewis Perin**
Date: 24 Sep 1993

A certain amount of hell's broken loose on LitProg since two of our favorite tool builders have "confessed" to what some regard as sins. Perhaps an analogy will convince the Calvinists that Briggs and Ramsey do deserve the respect they've long enjoyed here.

Think of literacy *outside* the realm of programming. We communicate with each other for many purposes, and to each purpose a certain amount of precision is appropriate. In informal conversation, if I tried to be as clear as I've hopefully been in some academic ventures, the person sitting next to me would stalk off impatiently before I got my first word off! Informal conversation isn't *worse* than the writing of a formal paper, it's just *different*.

Where I work, they're nice enough to pay me to write C++ "even though" I use CWEB. The webs are arranged for narrative coherence up to a point (I'd go crazy if I hadn't the ability to rearrange scraps), and there are plenty of grammatical English paragraphs free of the restrictions of C++ commenting, but none of it's publishable. Sure, this is a compromise, but a tenable one in my opinion.

As I remember, Kafka wrote reports for an insurance company by day, writing his lucid stories on his own time. Presumably his case reports were above average in clarity, but he'd have been fired quickly if he'd given them the care he devoted to his fiction.

From: **Mark Ng**
Date: 24 Sep 1993

I confess that my tool, noweb, is at best of poor quality when

You say that it is of poor quality but rest assured that it is being put to good use :)

i) Making a good-quality literate program is 3-4 times as expensive as just making a working program using literate-programming tools.

I could not agree more.

ii) I have been unable to develop really good literate programs without peer review.

I've used noweb for some time now and I enjoy using it because it is so simple. I went through most the different litprog tools and found noweb best suit to my task of multiple languages in one file. Thank you for your contribution of noweb as a tool :)

From: **Norman Ramsey**
Date: 24 Sep 1993

I wrote: I tolerate my ugly "literate" programs because in my work a program is seldom an artifact of the first importance. Far more

often the paper is what matters (and what gets polished).

And Chris Gray responds: Well that's a pretty straightforward admission that to this guy at least literate programming is just an academic fad which enables you to get papers published, not a serious proposal to enhance the quality of our programs (and our lives).

I think Mr. Gray is slightly misinformed about publication. The days of literate programming as a publishing fad have come and gone. Major publications are not amused by article about literate programming, and papers on literate program do not weigh heavily with those who play the publication game. So *I* wouldn't waste *my* time on literate programming if all I cared about was getting published.

I will now follow up with yet another variation on my standard polemic about literate programming. Longtime readers may want to skip it :-) The single, sad truth that Mr. Gray has unwittingly uttered is that after nine years literate programming is still at the proposal stage. It is *proposed* that literate programming will improve the quality of our programs. There's almost no evidence in favor of this proposition. The only cases I know of in which literate programming was used for production code and someone troubled to evaluate the results are Knuth's "Errors of TeX" and my "Literate programming on a team project" (with Carla Marceau).

There's little evidence that literate-programming tools can really give us better programs (although there are True Believers, of whom I am one). There is no method that tells one *how* to apply literate-programming tools to get good results. If I had the evidence and the method, I could take *any* of the existing tools to our development organization and put them into use. If I had only the method, I might find a project that would be willing to gamble on the results. I could never go to a project and say "here are some good tools; if you use them you will get good results." I have seen what damage novices can do with sharp instruments.

I am disappointed by the vast amount of work put into tools when what we desparately need is an attempt at "How (and how not) to write a good literate program no matter what tool you are using." There are endless implementations of tools and discussions of same because it's much easier to write tools and to evaluate tools than it is to teach people to program differently and to measure the quality of the results. Too bad.

From: **Zen**
Date: 24 Sep 1993

Preston Briggs writes: zen@death.corp.sun.com writes a lot of stuff. I won't quote it. He says I'm a hypocrite for writing nuweb in a less-than-perfectly-literate fashion. I disagree. I'm not preaching literate programming, I'm just a user.

As a clarification, I didn't say that you were hypocritical for writing a literate tool using non-literate methods. I said the *justification* you used seemed hypocritical to *my* ideal literate programming. I didn't say, nor am I trying to imply that nuweb is a poor program at all (I think the only thing I implied about nuweb is that it, like any program, would be better if written in a literal fashion; I suppose we would agree there, but that's not my issue.) Ok, just a couple of points; specifics in where we disagree:

The question of how it's coded is an entirely orthogonal issue. In fact, I'd say there are 3 orthogonal issues here [...] 1) The "language" definition -- the features (and misfeatures) of the system. 2) The actual collection of tools that support the language (in the case of web, the tangle and weave programs, and maybe the TeX processor and Pascal compiler). 3) How each of the tools is implemented.

Ideally, perhaps, features should be independent or implementation and tools used and available. I think, however, in practice, they are connected; I would say especially #3, and perhaps even more so #2, how they are implemented -- using literate programming or not -- is of extreme importance, *iff* you believe that it adds to the value of the program. In addition, when you use these tools to build other systems or tools, then it becomes even more important and non-orthogonal. I think that literate programming has less import in an ideal world than in the real world for those same issues you mention.

Nobody has commented on how I implement my state machines or how I parse command-line arguments or how I allocate memory. Why? Because those're just boring details. I'm not trying to teach people how to program, I'm trying to learn what makes a nice language for literate programming.

Perhaps people would comment on them, if they could read your code (and I don't mean just making it available to them.)

From: **Orion Auld**
Date: 25 Sep 1993

*zen@death.corp.sun.com writes: No less, of course. The point I was trying to make was that your justification was being hypocritical to the (*grin* -- well, *my* ideal, of course!) ideal of literate programming.*

You may be a literate programmer, but your degree of English literacy is underwhelming. The word "hypocritical" implies a contradiction of self, not a contradiction of someone else, and hence is usually regarded as being much more serious. Perhaps you meant to use "contradictory", in which case you would have to specify an object.

From: **Ross Williams**
Date: 26 Sep 1993

I've noticed someone suggesting that it is hypocritical to write a literate programming tool in a non-literate style. I disagree. Writing a literate tool using itself is just asking for trouble. What if the only executable is accidentally deleted?! Better to keep the implementation of the literate tool "clean". I wrote FunnelWeb in raw C on purpose. My attitude was: "One million billion non-literate programs have already been written. For the sake of implementation safety, just one more won't hurt." Of course, I haven't written an illiterate program since.

From: **Remo Dentato**
Date: 27 Sep 1993

*Norman Ramsey said: There's little evidence that literate-programming tools can really give us better programs (although there are True Believers, of whom I am one). There is no method that tells one *how* to apply literate-programming tools to get good results.*

I'm a True Believer too, and I agree with you. Now the problem is: there is such a method? As far I know there is no method to follow to write better novels, poems, and so on. Now I became a little mistic: writing code (literate or not), painting, composing music and other are a sort of illuminatio: you can show the results to others, you can try to expalain your Way to others, but you can't give them a recipe to follow. We could look for statical proof that literate programs are "better" (in some sense) but I don't know if there are enough test-cases.

I agree with you that the discussion should move to "How write literate programs", but I don't know how to start such a discussion! I know that you (and many others in this list) have a long experience in the literate programming field, could you (and others) suggest a starting point on the topic "How (and how not) to write a good literate program no matter what tool you are using."?

From: **Lee Wittenberg**
Date: 27 Sep 1993

Eitan Gurari writes: I consider it a worthless effort to polish the exposition and appearance of literate programs that are not intended to be consumed by others (i.e., documents that are not intended for peer review---see norman's note). Consequently, documents that serve as literate programs to me might be cryptic creatures to other readers.

I disagree. The time spent polishing exposition, in my experience, helps to find latent bugs. In addition, the better the web looks, the more likely I am to pass it around for an informal peer review (and the more likely others are to actually read it). I think a quote from Kernighan & Plauger's "Elements of Programming Style" might be in order here:

> One excuse for writing an unintelligible program is that it is a private matter. Only the original programmer will ever look at it, and surely he need not spell out everything when he has it all in his head. This can be a strong argument, particularly if you don't program professionally. It is the same justification you use for writing "qt milk, fish, big box" for a grocery list instead of composing a proper sentence. If the list is intended for someone else, of course, you had better specify what kind of fish you want and what should be inside that big box. But even if only you personally want to understand the message, if it is to be readable a year from now you must write a complete sentence. So in your diary you might write, "Today I went to the supermarket and bought a quart of milk, a pound of halibut, and a big box of raisins." You learn to write as if your someone else because next year you will be "someone else."

I must admit that I find even half-hearted webs (such as the source for noweb -- sorry, Norman) easier to read (and modify) than well documented & structured non-web code (such as the GNU project stuff) that I've attempted to play with in the past.

My webbing style seems to focus on the documentation rather than the code. I'm not a formal program-prover, but I try to use the text chunks (to use noweb terminology) to explain and "prove" the algorithm. The code chunks must match the documentation. I do this because I find it easier to reason in English than in C, Pascal, Awk, or any other non-human language. These days, when I find an error, it is usually because incorrect code does not match correct documentation, rather than the other way around. Adding fuel to the fire,

---

From: **Joachim Schrod**
Date: 27 Sep 1993

Norman Ramsey writes: I think Mr. Gray is slightly misinformed about publication. The days of literate programming as a publishing

*fad have come and gone. Major publications are not amused by article about literate programming, and papers on literate program do not weigh heavily with those who play the publication game. So *I* wouldn't waste *my* time on literate programming if all I cared about was getting published. George Greenwade writes: What would the group think of a "refereed electronic periodical" for literate programming; maybe "Literate Programming Review"*

Not much. The point is, IMNSHO, that the problems where LitProg gets interesting are too large to get published. In particular, we have found that LitProg is of great value if one has a middle sized system, i.e., something larger than 1MB WEB source size. Going above ~ 20 MB the LitProg paradigm gets irrelevant since other problems plague you more. (Yes, we wrote a system in this size. And we wrote some smaller systems with approximately 5 MB of code.) How do you want to publish that?

Another point: LitProg is very nice in a group where you have a high fluctuation of people working on a project. (Like we have here at the university.) The positive effects of LitProg are in the maintainance phase, not in the development phase. There lies also the problem with Norman's demand of empiric results: Who has the resources to create a controlled experiment that has to last for at least two years with a lot of people? We surely don't. Not that it would matter, IMO: The journals flourish with papers about methods where nobody even bothered to use them in toy projects. Our problem is more that we must introduce more formulisms...

By god, you're fast. I wrote: has a middle sized system, i.e., something larger than 1MB WEB source size. Going above ~ 20 MB

To save myself from further questions: I would not even *think* about writing a source file of 1 MB. I meant total size of WEB documents, whereby I count specifications and implementations. Documents concerning analysis, design, test, configuration, and the (iterative) software process itself are not included in this figure.

From: **Eric van Ammers**
Date: 28 Sep 1993

Norman Ramsey writes: There's little evidence that literate-programming tools can really give us better programs (although there are True Believers, of whom I am one).

I'm a true beleiver too and I'm slightly less pessimistic here. The work of Oman is at least a hint at the value of literate programming. 1) Oman, Typographical style is more than cosmetic, CACM 33, 5, 506-520. 2) Oman, the book paradigm for imporved maintenance, IEEE software 1990 pg 39-45.

From: **Remo Dentato**
Date: 28 Sep 1993

I've noticed someone suggesting that it is hypocritical to write a literate programming tool in a non-literate style. I disagree. Writing a literate tool using itself is just asking for trouble. What if the only executable is accidentally deleted?! Better to keep the implementation of the literate tool "clean".

I strongly disagree! I think that a literate programming tool should be the first example of using the tool itself. The problems of accidentally erase your executable is a matter of organization. I've written a nuweb-compatible litprog tool that use only ascii (I've not finished the formatting part, but the generation of code is very fast and uses very little memory: I've to work on micros too :-)). Anyway, it was the first program written using itself and I had no problems of loosing executables or such.

From: **C.M. Sperberg-McQueen**
Date: 28 Sep 1993

In the discussion over whether literate-programming tools should or should not be implemented in the languages they define, I think the humility of Preston Briggs and Norman Ramsey may have allowed a misconception to take root. Nuweb, at least, is in fact implemented in nuweb, and provides a nice little test of literate programming tools: is the program easier to understand even when the document is still incomplete? (Remember, this is version 0.8!)

Nuweb passes this test with flying colors. Even though the commentary becomes rather sparse during the chapter on The Overall Structure, and mostly non-existent in the chapter on The Support Routines, nuweb is still more legible than the other C programs of comparable size and complexity that I have come into contact with, because of the cross-referencing between scraps of code and because of the index.

I can make this comparison with some confidence, because I just finished porting nuweb to VM/CMS, in about three hours of work (including probably forty-five minutes wasted trying to remember how the debugger works, and interrupted by several hours during which my subconscious mulled over the problems, and eventually realized that ASCII is a seven-bit and EBCDIC an eight-bit character set). This compares favorably with other software I have ported, which has taken days or (in one case, where the thing

never did actually run) weeks. Of course, this could be because Preston Briggs writes cleaner code, but perhaps he does that because he's using a web system.

While I agree that more commentary would make nuweb a more readable program, I disagree firmly with those who claim it isn't literate, in any useful sense, now. If anyone else can use a VM/CMS version of nuweb, they should let me know.

From: **Kevin Cousins**
Date: 29 Sep 1993

Eric van Ammers writes: 1) Oman, Typographical style is more than cosmetic, CACM 33, 5, 506-520. 2) Oman, the book paradigm for imporved maintenance, IEEE software 1990 pg 39-45.

Nice articles. If anyone on LitProg hasn't already checked them out, get them now. Honestly, trying to write code that uses this book paradigm can get to be a real pain in the ass. To discover an entire battery of tools available (ala LitProg) is a real blessing, despite all the recent arguments about their relevance.

Luminary abuse

From: **Norman Ramsey**
Date: 22 Oct 1993

["chunk"] is perhaps the kind of word (like 'memory' and 'flavor') bound to meet with the disapproval of Edsger Dijkstra

What stronger endorsement could anyone ask for?

From: **Dave Mason**
Date: 24 Oct 1993

["chunk"] is perhaps the kind of word (like 'memory' and 'flavor') bound to meet with the disapproval of Edsger Dijkstra

Norman Ramsey writes: What stronger endorsement could anyone ask for?

Wow, one week Preston says that Knuth doesn't write readable webs, and the next week Norman implies that any putative anti-endorsement by Dijkstra is good enough for him! What can we say bad about Tony Hoare, Nicholas Wirth, Alan Kaye, Al Aho or Ken Thompson next week? :-) (apologies to any luminaries I forgot to nominate for abuse :-)

I'm no-one to talk, but I tend to agree with Preston. (Knuth is brilliant, but I believe TeX-the-Program and Metafont-the-Program are significant dis-incentives to the spread of literate programming. Virtually anyone I've tried to interest in literate programming who has seen TtP uses it as a counter to my arguments. This may be an argument that TtP is outside the domain of program sizes for which web works well -- maybe hypertext web would be better. I'm not sure the programs Knuth wrote for Jon Bentley's column are completely convincing either -- I think they are perhaps too clever and didactic.)

While Dijkstra has many extreme ideas, I think anybody dismisses him at their intellectual peril. My experience is that ideas like literate programming mesh very well with Dijkstra's ideas on program development. (I'm not implying that Norman was dismissing him.) In general I think ritual luminary abuse is probably not very constructive.

From: **Matthias Neeracher**
Date: 24 Oct 1993

Dave Mason writes: Wow, one week Preston says that Knuth doesn't write readable webs, and the next week Norman implies that any putative anti-endorsement by Dijkstra is good enough for him! What can we say bad about Tony Hoare, Nicholas Wirth, Alan Kaye, Al Aho or Ken Thompson next week?

This is a little off-topic, but how about Alan Perlis et al. abusing Tony Hoare? "C.A.R. Hoare has been quoted as saying, `In many applications, algorithm plays almost no role and certainly presents almost no problems.' (We wish we could report that he thereupon threw up his hands and abandoned verification, but no such luck.)" -- DeMillo, Lipton, and Perlis, _Social Processes and Proofs of Theorems and Programs_

I'm no-one to talk, but I tend to agree with Preston. (Knuth is brilliant, but I believe TeX-the-Program and Metafont-the-Program are significant dis-incentives to the spread of literate programming.

I disagree. While they IMHO might profit from some more global documentation (maybe a diagram of where the various parts fit in), I still believe that they are excellent examples of literate programming.

While Dijkstra has many extreme ideas, I think anybody dismisses him at their intellectual peril.

My problem with Dijkstra is that he pontificates about topics that he appears to have less and less experience with. Does anybody here happen to know when Dijkstra last wrote *himself* a program of any substantial size? I would assume that he has not done so for at least 15 years. Dijkstra is bound to generate resentment with people actually working with computers when he declares that advanced debuggers are unnecessary and harmful, and some system designers actually take him seriously, or when he declares that he does not have time to use word processors and writes everything with a fountain pen (As he did in a CACM article; I don't remember him having made any mention of his Secretary in that article).

My experience is that ideas like literate programming mesh very well with Dijkstra's ideas on program development.

This is probably true, as literate programming is very well suited to stepwise refinement and having comments typeset is very advantageous if you want to do formal reasoning on the program.

In general I think ritual luminary abuse is probably not very constructive.

I'm not so sure of that. Dijkstra himself seems to have built a considerable part of his reputation on polemics (I mean, who would remember him without "goto statement considered harmful") and has never stopped short of abusing his colleagues (Dijkstras "Selected Writings on Computing: A Personal Perspective" contains a few remarks so nasty that the names of the targets had to be removed out of legal concerns).

From: **Steve Avery**
Date: 25 Oct 1993

Dave Mason says: Knuth is brilliant, but I believe TeX-the-Program and Metafont-the-Program are significant dis-incentives to the spread of literate programming.

Sure, but you must admit that the original Web for Pascal was a bit primitive (as probably was the version of Pascal that was used). In all fairness, I think it would be wise to wait for Knuth's next book (the precursor to Volume 4) which apparently has literate programs in it, hopefully using CWEB. p.s. And you can't cite CWEB as an example of Knuth's poor programming, as most of it isn't his (and I don't think its that bad an example of programming either).

From: **Norman Ramsey**
Date: 25 Oct 1993

Wow, one week Preston says that Knuth doesn't write readable webs, and the next week Norman implies that any putative anti-endorsement by Dijkstra is good enough for him!

Actually I have the greatest respect for Dijkstra's work on program correctness and the calculus of guarded commands. I also, however, have the greatest amusement for his handwriting fetish and other eccentricities. Doubtless I will never make a formalist.

What can we say bad about Tony Hoare, Nicholas Wirth, Alan Kaye, Al Aho or Ken Thompson next week? :-) (apologies to any luminaries I forgot to nominate for abuse :-)

Now wait a minute! Tony Hoare is my hero.

I'm no-one to talk, but I tend to agree with Preston.

Me too (all kidding aside). The details of character translation are much less interesting than the line-breaking and other algorithms.

I'm not sure the programs Knuth wrote for Jon Bentley's column are completely convincing either -- I think they are perhaps too clever and didactic.)

I find them completely convincing for what they are --- I think MacIlroy used the phrase "industrial-strength Faberg\'e egg." Impressive and repays careful study. I am eagerly looking forward to the upcoming book.

While Dijkstra has many extreme ideas, I think anybody dismisses him at their intellectual peril. My experience is that ideas like literate programming mesh very well with Dijkstra's ideas on program development. (I'm not implying that Norman was dismissing him.)

The disappointing thing about those ideas is that a) they haven't been advanced, changed, or enriched in 15 years b) they are tremendously difficult to apply beyond toy examples In my travels I have met one programmer who uses Dijkstra's methods to write serious software---and this is a guy who is brilliant and has made original contributions to program correctness as well as other areas. This is not to say Dijkstra's methods are useless to mere mortals---I use them every time I write a loop---but to say that they have never delivered on their grand promises.

In general I think ritual luminary abuse is probably not very constructive.

But I'm having such fun! OK, so we're far afield from literate programming. I promise to keep my lip buttoned from here on.

From: **Joachim Schrod**
Date: 25 Oct 1993

Matthias Neeracher writes: Dijkstra himself seems to have built a considerable part of his reputation on polemics (I mean, who would remember him without "goto statement considered harmful")

At least, I would. For his work on structured programming, his work on multi-processing, his work on compiler construction, on the formal definition of programming languages, on the essence of control structures, on non-determinism, and, and, and. In addition, I can't find polemics in his CACM letter. It was a well reasoned statement about the consequences of the (ab)use of certain control structures. Go ahead and read it again! The discussion afterwards was heated and filled with polemics, but not the letter.

Yes, he's a very arrogant person and he seems not to be in touch with reality often enough -- but he's brilliant, too. Concerning Literate Programming and Dijkstra: Knuth himself acknowledges the influence of the paradigms of Dijkstra on his ideas of programming style.

From: **Mike Yoder**
Date: 25 Oct 1993

Matthias Neeracher writes: Dijkstra himself seems to have built a considerable part of his reputation on polemics (I mean, who would remember him without "goto statement considered harmful")

A fact which may be relevant in this context: Dijkstra did not choose this title; it was placed over what had been simply a letter to the editor, as it were, by the publisher. I have read the letter and don't remember its contents as being particularly polemical; perhaps that is because I have seen the claims about the potential harm of gotos borne out both in professional and in academic contexts. Others may disagree, but in my judgment saying that X is harmful isn't polemical when X is, in fact, harmful; nor is it usually polemical even when X isn't harmful. The speaker may, after all, be mistaken.

From: **Lee Wittenberg**
Date: 25 Oct 1993

Dave Mason writes: Wow, one week Preston says that Knuth doesn't write readable webs, and the next week Norman implies that any putative anti-endorsement by Dijkstra is good enough for him! What can we say bad about Tony Hoare, Nicholas Wirth, Alan Kaye, Al Aho or Ken Thompson next week? :-) (apologies to any luminaries I forgot to nominate for abuse :-)

I have to admit that I thought about making the same kind of crack about Dijkstra, myself; I'm glad Norman beat me to it :-). Seriously though, Dijkstra does tend to take extreme positions. I like to believe that he does it primarily to start discussion, rather than because he actualy advocates the extremes, but I tend to think the best of people :-).

I'm no-one to talk, but I tend to agree with Preston. (Knuth is brilliant, but I believe TeX-the-Program and Metafont-the-Program are significant dis-incentives to the spread of literate programming. Virtually anyone I've tried to interest in literate programming who has seen TtP uses it as a counter to my arguments. This may be an argument that TtP is outside the domain of program sizes for which web works well -- maybe hypertext web would be better. I'm not sure the programs Knuth wrote for Jon Bentley's column are completely convincing either -- I think they are perhaps too clever and didactic.)

I agree with Preston and Dave on this one, but let us not forget that TeX the Program was Knuth's ~first~ literate program. Remember your own early attempts at literate programming, and look at how far you've come since then. I'm sure that if Knuth were to write TeX now, the exposition and layout would be totally different. Of course, Knuth does have a Turing-like streak that occasionally substitutes cleverness for clarity, but perhaps that is one of the marks of greatness.

In general I think ritual luminary abuse is probably not very constructive.

Agreed, but it is fun.

From: **Chris Gray**
Date: 26 Oct 1993

Matthias Neeracher writes: Dijkstra is bound to generate resentment with people actually working with computers when he declares that advanced debuggers are unnecessary and harmful, and some system designers actually take him seriously,

Hear hear hear hear hear. The system on which I have spent half of my working life started out with that kind of thinking---no need for online debugging, everything will be ``finite message machines'' individually tested beforehand,... several years later, the debugging stuff was all there, but not until the project had soaked up enough unbudgeted man- power to bring one of the world's great multinationals to its knees.

Multi lingual literate programming

From: **Evan Kirshenbaum**
Date: 22 Sep 1993

I would very much like to experiment with WEB-style (or otherwise) literate programming. Unfortunately, my programs tend to be written in several languages at once. As an example, my most likely first candidate is a second generation of a document formatting tool, and will include: the formatter, written in C++, a set of PostScript functions used by the PostScript code output by the formatter, Emacs-lisp functions to allow easy specification of the formatter input, a (default) configuration file in a format to be specified by me, a man page written in (ugh!) nroff, and (probably) a few perl scripts.

One of the attractions of programming in expositional order is the ability to have a section on, say, line breaking, in which I can include the code to process relevant arguments, the code to break the lines, the PostScript support routines that deal with continuation lines, and the man page entry documenting these arguments. As it stands now, the code is distributed throughout several files. I know that there are WEB-like tools that allow you to customize the programming language, but does anyone know of a tool in which each different code fragments can be in different languages?

From: **John Krommes**
Date: 22 Sep 1993

Evan Kirshenbaum writes: I know that there are WEB-like tools that allow you to customize the programming language, but does anyone know of a tool in which each different code fragments can be in different languages?

FWEB features multiple language support, and one can easily switch between languages within a single web source file. Presently the supported languages are C, C++, Fortran-77 and Fortran-90, Ratfor-77 and Ratfor-90, and TeX. I am completing work on a language-independent mode. If you are very interested in that feature, please correspond with me privately; the speed with which that project is completed will depend on the number of requests.

From: **Osman Buyukisik**
Date: 22 Sep 1993

Try nuweb, noweb. The first is one executable. noweb consists of a number of programs. I think for a beginner nuweb is easier to learn (just a few commands). In either case you need to know a little LaTex or TeX. Yeah, Funnelweb is also another choice but a few more commands (an TeX only I think), and with a much bigger manual than the either of the first two. I think all can or already are ported to quite a few platforms including DOS.

From: **Lutz Prechelt**
Date: 22 Sep 1993

Osman F Buyukisik writes: In either case you need to know a little LaTex or TeX. Yeah, Funnelweb is also another choice but a few more commands (an TeX only I think), and with a much bigger manual than the either of the first two.

You can use FunnelWeb with LaTex, too. Just say \def\bye{\relax} at the beginning of your FunnelWeb document and there you go. (To use 11pt or 12pt style, you have to redefine a few commands) FunnelWeb is quite powerful (e.g. parameterized macros). The manual is really very large, but that does not mean that you cannot get started within 10 minutes with FunnelWeb: the basics are quite similar to the way the simple Webs like NoWeb work.

From: **Norman Ramsey**
Date: 22 Sep 1993

I would very much like to experiment with WEB-style (or otherwise) literate programming. Unfortunately, my programs tend to be written in several languages at once... does anyone know of a tool in which each different code fragments can be in different languages?

noweb was designed precisely for this purpose (and for simplicity). A stable version is available via ftp from ftp.cs.princeton.edu:pub or from bellcore.com:pub/norman. An alpha test version is available from me by email. The alpha version adds many improvements to

the LaTeX output, language-independent identifier cross-reference and indexing, and bugs. Lots of bugs.

From: **Lee Wittenberg**
Date: 22 Sep 1993

Evan Kirshenbaum writes: I would very much like to experiment with WEB-style (or otherwise) literate programming. Unfortunately, my programs tend to be written in several languages at once. As an example, my most likely first candidate is a second generation of a document formatting tool, and will include: the formatter, written in C++, a set of PostScript functions used by the PostScript code output by the formatter, Emacs-lisp functions to allow easy specification of the formatter input, a (default) configuration file in a format to be specified by me, a man page written in (ugh!) nroff, and (probably) a few perl scripts.

One of the attractions of programming in expositional order is the ability to have a section on, say, line breaking, in which I can include the code to process relevant arguments, the code to break the lines, the PostScript support routines that deal with continuation lines, and the man page entry documenting these arguments. As it stands now, the code is distributed throughout several files. I know that there are WEB-like tools that allow you to customize the programming language, but does anyone know of a tool in which each different code fragments can be in different languages?

noweb can definitely handle the job. I believe that nuweb and FunnelWeb will also do the trick, as will CLiP. I, myself, have written a single noweb web (awkward phrase, but I couldn't think of a better one) that contains a C program and a DOS batch file that invoked it. Jon Krom's cross indexer for noweb contains a Unix shell script, a DOS batch file, and an Awk script all in one web. The main point here is that what you want is eminently doable with already existing tools.

Readable Programs - an Alternative to Web

From: **Stephen Savitzky**
Date: 23 Sep 1993

My approach to literate programming is to write programs that can be read, rather than the Web approach of writing programs that can be processed to make them readable. I do this because I rarely, if ever, get a listing; I much prefer to edit on the screen. I want my programs to be as readable as possible when I do.

My coding standards for readable programs are: Comments that describe a single code unit such as a declaration or statement *follow* that code unit, indented as if they were a statement continuation. (Comments describing a C++ class or C structure declaration follow the first line.) This has the effect of making the declaration a header for the comment rather than the other way around. Therefore, comments need contain no information that is already in the code. This avoids problems with comments and code getting out of sync. In particular, comments must not contain either a rephrasing of the declaration they describe, or any cross-reference information that is derivable from the code. Comments describing or summarizing a group of code units *preceed* the members of that group.

I *do* have a (TeX-based) typesetter for C++ programs that uses a set of very simple rules; the typesetter is only used for header files, and the rules produce something that can be used as a programmer's manual for the modules described by the header.

The typesetting rules are: Code is printed in a monospaced font to preserve its formatting. Comments are printed in a proportionally-spaced font, and may be paragraph-wraped. Any indentation to the left of them, however, is printed with the same spacing as code. Comments consisting of a line of comment-start characters print as horizontal rules, and their first line is used as a section header. Any comment line ending in ":" is boldfaced to serve as a subsection head. Code that appears to be function bodies (i.e., in C, code that starts with a left brace in the left-hand column, plus inlined functions in C++) is omitted. There is a distinctive set of comment delimiters that cause anything between them to be omitted. These are used to keep kludgery out of the user-level documentation.

```
Example:

class Point {
        // Description:
        //        a trivial example -- C++ version of a point in polar
        //        and cartesian coordinates.

        // Private data:

        float x_, y_;
```

```
        // Cartesian coordinates

        float x();
        float y();

        // Polar coordinates:

        float rho();
        // Radius.
        float theta();
        // Angle.
};
```

From: **Osman Buyukisik**
Date: 23 Sep 1993

Here is a short reply: IMHO you are missing a lot if you just rely on commenting. Just try nuweb or noweb. You also get to arrange your code in a more understandable way (good for during the initial design phase). Need to live with LaTeX though! I like the printed stuff but in your case if you are using DOS/Windows try WinWordWEB, this is a WYSIWYG system. May be others are working on a unix/X system. This way you don't have to make up a strict commenting system and remember how to use it!

From: **Marcus Speh**
Date: 24 Sep 1993

Stephen Savitzky said: My coding standards for readable programs are: [..very interesting stuff deleted..]

Hey, this is another interesting article--as far as I know the first time that the word of "coding standard" comes up in connection with literate programming, though we've been discussing litprog 'habits' at length in the past [I checked with the archives through gorgeous gopher search, only one entry found which refers to pretty-printing of fortran output using sed(1) :-)]. I have often thought this is something we should have-- I (and many of my colleagues) did profit from such a document for c++ (called RULES AND RECOMMENDATIONS --> ftp.desy.de:/pub/c++/misc/c++.rules.ps), though any standardization document is (and should) usually subject to heated discussion.

Now, is it heresy to think of a "coding standard", as a set of minimal rules for the practicing literate programmer? Clearly, such a text should be independent of the particular litprog tool--for the same reason it may support unifying tendencies, which I would welcome.

Big Programs & Make & Incremental Compilation & Web

From: **Robert McLay**
Date: 23 Sep 1993

I would like to start a discussion about web's and big projects. I freely admit to be a complete newbie when it comes to webs but my reading seems to say that programs in web tend to be much bigger that you might typically see in a unix style development. I believe that tex program is stored in 6-10 files. What I want to know is how well web's work when the programs are say 20,000 to 100,000 lines of code. If I make a small change in the program I would certainly like not to have to recompile all of the code.

Now I freely admit that a program in web will be less error prone, so I might need less re-compiling but I still like quick re-compiles possible with make. So one way to answer my question is which of the webs handle a program spread over many files and are easily coupled with make. Another question is how well have people handled several people working on the same project w/ web.

From: **Mike Yoder**
Date: 23 Sep 1993

Quoting Robert McLay: What I want to know is how well web's work when the programs are say 20,000 to 100,000 lines of code. If I make a small change in the program I would certainly like not to have to recompile all of the code.

There is a simple method which accomplishes this and even more: if you change only the documentation, you need not recompile anything. As your programming scale goes up, this feature gradually changes from being pleasant to being essential. In order to avoid elaborate circumlocutions, I will describe the particular system I use rather than a general one. **Inscribe** generates some number of files when it is run over its source (in what WEB users would call tangle mode, though this verb isn't appropriate here). For the sake of

concreteness say they are m1.h through m9.h and m1.c through m9.c.

Logically, what happens is that new versions of m1.h etc. are generated and compared to their corresponding old versions. If two such are identical, the new one is deleted, leaving the old one untouched; otherwise it replaces the old one. On UNIX, this is done literally (the new files are named m1.h+ etc.); on VMS, the new file is a more recent generation of the old one. (On UNIX, the old one is renamed m1.h~ to provide 1-level backup of old versions, but this isn't essential to the method.) The critical thing is that if m1.h doesn't change, it isn't touched, so a "make" won't trigger recompilations. A naive implementation, obviously, would generate completely fresh versions of all generated files, and so any change at all would require "recompiling the world."

A side effect of this is that just typing "make" isn't enough; you must first run Inscribe, then make. This doesn't bother me because I think the make file should itself be made from the Inscribe source, but some have considered this a negative aspect. (I have also observed that nearly all large software projects use some kind of makemake facility layered on top of make, so this "misfeature" is not likely to matter.) This approach has the advantage that you may be able to layer it on top of existing literate programming tools via scripts. I recommend it highly.

From: **Preston Briggs**
Date: 23 Sep 1993

Most web tools support the creation of multiple output files (certainly nuweb, noweb, CWEB 3.0, FWEB, FunnelWeb). That is, within a single web, I can specify many files of code that will be created by the web tool. Many (all?) of these tools have a provision that avoids overwriting a code file if it would be unchanged. Thus, it's easy to avoid massive recompiles when using make. Avoiding big TeX costs is harder. I'd like to modify nuweb to take advantage of the Latex \include mechanism, but haven't done so yet.

Multi-person projects are harder to coordinate. In our case, we work on a multipass compiler, so there's a fairly natural division of labor and we have divided the project into a number of webs, one per pass, with a single overview/coordinating web that describes the common elements of each pass. I'm not sure how best to arrange more complex programs.

From: **Lewis Perin**
Date: 23 Sep 1993

[...] I would certainly like not to have to recompile all of the code. Mike Yoder writes: There is a simple method which accomplishes this and even more: if you change only the documentation, you need not recompile anything. As your programming scale goes up, this feature gradually changes from being pleasant to being essential. [details of exactly how he does it]

Unless I'm missing something this doesn't conquer the problem. Those of us who don't get their code right the first time (disgusting wretches we, failing even with literate programming tools;-) tend to rely on our tangles inserting hints to the target language debugger as to where in the web the source code can be found. Trouble is, change the documentation significantly and the locations will change. If the source code hasn't changed but has been relocated in the web, a tangle smart enough to notice this can either go ahead and fool make (good) and the debugger (bad), or it can let the compilation steps cascade. Right?

From: **Joachim Ziegler**
Date: 24 Sep 1993

*Mike Yoder writes: **Inscribe** generates some number of files when it is run over its source (in what WEB users would call tangle mode, though this verb isn't appropriate here).*

Can anyone please explain me what "Inscribe" is?

From: **Mike Yoder**
Date: 24 Sep 1993

Lewis Perin wrote, in response to my deathless prose: Unless I'm missing something this doesn't conquer the problem. Those of us who don't get their code right the first time (disgusting wretches we, failing even with literate programming tools;-) tend to rely on our tangles inserting hints to the target language debugger as to where in the web the source code can be found. Trouble is, change the documentation significantly and the locations will change. If the source code hasn't changed but has been relocated in the web, a tangle smart enough to notice this can either go ahead and fool make (good) and the debugger (bad), or it can let the compilation steps cascade. Right?

You did miss something, but only because I omitted to tell you. Inscribe automatically inserts #line directives for generated C and C++ files, so the debugger's source is the Inscribe source in those cases; I call this "direct debuggability," which may be my private jargon, but I think I appropriated the term from people in this forum. It would be simple to generate equivalent directives for other languages if their compilers paid attention to them, e.g. "pragma Line" for an Ada compiler. Also, I have an Emacs function which lets me find the Inscribe line which corresponds to line N of a given produced file: thus if the compiler says I have an error on line 32 of Pascal file foo.p, I find some occurrence of source for foo.p and invoke "C-Z g" with an argument of 32, and it goes to the appropriate place. This is crude but quite effective and reliable.

From: **Lewis Perin**
Date: 24 Sep 1993

Mike Yoder writes: You did miss something, but only because I omitted to tell you. Inscribe automatically inserts #line directives for generated C and C++ files, so the debugger's source is the Inscribe source in those cases; I call this "direct debuggability," which may be my private jargon, but I think I appropriated the term from people in this forum. It would be simple to generate equivalent directives for other languages if their compilers paid attention to them, e.g "pragma Line" for an Ada compiler.

OK, Inscribe does what CWEB, FWEB and noweb (at least) do in this respect. Sorry, but I still don't see how this gets around the dilemma of code relocation within the web mentioned in the second paragraph of my original posting.

Also, I have an Emacs function which lets me find the Inscribe line which corresponds to line N of a given produced file: thus if the compiler says I have an error on line 32 of Pascal file foo.p, I find some occurrence of source for foo.p and invoke "C-Z g" with an argument of 32, and it goes to the appropriate place. This is crude but quite effective and reliable.

Yes, but will the *debugger* find the code if the latter's been relocated without being recompiled?

From: **Frank Jensen**
Date: 24 Sep 1993

Lewis Perin wrote, in response to my deathless prose: Unless I'm missing something this doesn't conquer the problem. Those of us who don't get their code right the first time (disgusting wretches we, failing even with literate programming tools;-) tend to rely on our tangles inserting hints to the target language debugger as to where in the web the source code can be found. Trouble is, change the documentation significantly and the locations will change. If the source code hasn't changed but has been relocated in the web, a tangle smart enough to notice this can either go ahead and fool make (good) and the debugger (bad), or it can let the compilation steps cascade. Right?

Mike Yoder replied: You did miss something, but only because I omitted to tell you. Inscribe automatically inserts #line directives for generated C and C++ files, so the debugger's source is the Inscribe source in those cases; I call this "direct debuggability," which may be my private jargon, but I think I appropriated the term from people in this forum. It would be simple to generate equivalent directives for other languages if their compilers paid attention to them, e.g. "pragma Line" for an Ada compiler.

The original poster meant something different: he already has a tool that inserts such #line directives in the tangled output files (C source files). The problem is that if you have one relatively large web file that generates a number of C source files and you change something near the beginning of the web file, then all the output files will change (because the line numbers in the #line directives change). If you ignore this type of difference when you compare files, then Make will not recompile the files, but the debugger will be confused because it relies on the #line directives to identify offending code. Or you don't ignore the difference, i.e., all files will compare unequal and recompilation will take a long time.

The real source of the problem is that the web is contained in one file. The way I solve this problem is to have one web file that includes a lot of small web files. Each of these small web files corresponds to one C source file. Thus, if I edit one of the small web files, it won't affect the line numbers of all the other files, and recompilation will be fast and the debugger will not be confused.

LitProg Review

From: **George Greenwade**
Date: 27 Sep 1993

*Norman Ramsey posted: I think Mr. Gray is slightly misinformed about publication. The days of literate programming as a publishing fad have come and gone. Major publications are not amused by article about literate programming, and papers on literate program do not weigh heavily with those who play the publication game. So *I* wouldn't waste *my* time on literate programming if all I cared about was getting published.*

[As if I didn't already have enough to do with network stuff]. I've been meaning to ask this for some time now of this group (for lurkers, news newbies, and others who are unaware of it, while I proudly "own" the LitProg list and you are sure to see my e-mail address more than you probably would/will want, I am generally clueless about programming; I'm an economist who enjoys the network).

What would the group think of a "refereed electronic periodical" for literate programming; maybe "Literate Programming Review" or some other less mundane name? What I have in mind is a model we are developing for economics -- you have the usual cadre of people, an editor, etc., with a panel of blind reviewers; articles are submitted via e-mail to the editor, who then strips author-identifying items, and forwards the submissions to a subset of the reviewers; the reviewers comment on the submission to the editor, who in turn forwards blind (stripped) reviews to the author; the author makes modifications, resubmits, etc. Once the reviewers and editor say so, the article is ready for inclusion in a mail-based "journal" (which can easily be linked into c.p.l). The address for the journal is

completely private for posting -- only selected addresses (say, only the editor) can post to it -- but completely public for subscriptions. An alias (or the direct editor's e-mail address) can be used for submissions.

As I see it, this would accomplish at least two things now missing and sorely needed. First, Norman's "Major publications are not amused by article about literate programming", is removed as a major journal, devoted exclusively to literate programming, would exist. It would be the journal for this not-so-limited topic. Insofar as whether or not it is a "journal", getting a unique ISSN for it is not a concern. Second, and IMO more important, it would have the ability to provide examples (where literate programming is sorely lacking) and discussions of real life constraints in both the philisophy as well as application of this concept of programming, with reference to a specific dimension covered in the article.

The electronic forum is an excellent place for this, IMO. First, it will be a long time before any major print periodical develops (or for a dedicated space for the topic within an existing print periodical is created) as I don't think a lot of people even understand the concept of literate programming (while I admittedly am not a programer, my understanding of its benefits makes me wonder why it is not more applied than it appears to be; I often -- possibly mistakenly -- think that I have a better conceptualization of literate programming than many professional programmers out there who could really benefit from it). Until the concept is spread, this form of print-related distribution is just about out of the question; without a distribution of some sort, the concept can't be spread -- sort of a serious Catch-22.

Second, the costs of production and distribution (which, as a former editor of a few print jourals, I can assure you are not trivial), as well as the costs of start up are minimized (rapidly approaching zero). Third, space constraints, such as number of physical pages per issue, simply do not exist in the electronic medium. Fourth, and related to the items above, periodicity is merely a side issue -- if a monthly format is used, with one or 200 articles or a statement that "nothing is included this month", so be it; the underlying concern is that the number of issues and when to expect their distribution are basically fixed in nature.

I'm not offering to edit this, but I am more than willing to provide a few resources to it (sort of serve as publisher, I guess) and discuss what can and (IMO) should be considered in doing this. It's a real opportunity for the enhancement of literate programming to maybe gain wider acceptance. If anyone is interested, let me know.

From: **Lee Wittenberg**
Date: 27 Sep 1993

George Greenwade suggests: What would the group think of a "refereed electronic periodical" for literate programming; maybe "Literate Programming Review" or some other less mundane name? What I have in mind is a model we are developing for economics -- you have the usual cadre of people, an editor, etc., with a panel of blind reviewers; articles are submitted via e-mail to the editor, who then strips author-identifying items, and forwards the submissions to a subset of the reviewers; the reviewers comment on the submission to the editor, who in turn forwards blind (stripped) reviews to the author; the author makes modifications, resubmits, etc. Once the reviewers and editor say so, the article is ready for inclusion in a mail-based "journal" (which can easily be linked into c.p.l). The address for the journal is completely private for posting -- only selected addresses (say, only the editor) can post to it -- but completely public for subscriptions. An alias (or the direct editor's e-mail address) can be used for submissions.

I think this would be a wonderful idea (although I was under the impression that someone ~was~ already working on similar lines). I would be happy to volunteer my services as a reviewer (without my contact lenses, I am legally blind :-), but not as an editor (sorry, I have very few organizational skills, and no desire to acquire any more). I would also certainly contribute the odd article, whether it was a "respected journal" or not. I think it can be an important forum for literate programming (although I also think that this mailing list/newsgroup is an important forum in and of itself).

From: **Norman Ramsey**
Date: 27 Sep 1993

What would the group think of a "refereed electronic periodical" for literate programming; maybe "Literate Programming Review" or some other less mundane name?

Short answer: it would be a super idea if you got some real heavyweights on the editorial board. People like Bentley, Van Wyk, or Hanson. The actual editor who does the dirty work needn't be such a heavyweight, but without some such backing I don't think it would be taken seriously outside a very small group. I don't know where you would find an editor who would be willing to put time into making it work. I can ask around...

From: **Christian Lynbech**
Date: 28 Sep 1993

Short answer: it would be a super idea if you got some real heavyweights on the editorial board. People like Bentley, Van Wyk, or Hanson. The actual editor who does the dirty work needn't be such a heavyweight, but without some such backing I don't think it would be taken seriously outside a very small group.

It depends much upon what you want to acheive. If academic fame is required, only the heavyweights will do, but if the emphasis is more on producing better programs, any board of knowledgeable and experienced LitProgrammers (or even just) programmers should

suffice. But as someone else (Joachim Schrod?) has mentioned, the real significance comes for real projects, i.e. large examples of thousands of lines. And I would very much doubt that anybody would want to go into that game, unless being part of the development itself, as seen in companies using reviews to produce better code.

But from an educational point of view, it may make sense after all. People experimenting with LitProgramming (or using it for small 1 person projects), could have their code reviewed, and perhaps published in whatever fora. This would give all contributors valuable critique and evaluation, and provide everyone with a body of nice, reviewed works, to learn from.

I don't know where you would find an editor who would be willing to put time into making it work. I can ask around...

I definitely like the idea. Perhaps it even applies to the Usenet University (not that I no much about that).

From: **Marcus Speh**
Date: 28 Sep 1993

Christian Lynbech said: But from an educational point of view, it may make sense after all. People experimenting with LitProgramming (or using it for small 1 person projects), could have their code reviewed, and perhaps published in whatever fora. This would give all contributors valuable critique and evaluation, and provide everyone with a body of nice, reviewed works, to learn from. I definitely like the idea. Perhaps it even applies to the Usenet University (not that I no much about that).

I can speak for Usenet University since I am on its board of directors. We are currently discussing the curriculum for our first semester on the internet (spring 94)-- we would certainly be honored to host a LitProg Review enterprise, though I can't say whether we can allocate any manpower for it.

We would like to bundle any sort of "virtual" educational activity -- a journal which is unlikely to ever never appear on real paper fits in this category. We already have: a list of electronic consultants, a collaborative (Hyper-) textbook project, an internet meta library, and a virtual campus. To offer a beginner's course on LitProg would be another nice course offer. Any help/suggestion is welcome. I will bring up the Review issue on the next board meeting on Thursday [a virtual meeting on MediaMOO at MIT].

From: **Eric van Ammers**
Date: 29 Sep 1993

We would like to bundle any sort of "virtual" educational activity -- a journal which is unlikely to ever never appear on real paper fits in this category. We already have: a list of electronic consultants, a collaborative (Hyper-) textbook project, an internet meta library, and a virtual campus.

I like the idea very much, but my main problem is the following. Scientists in a university environment are (at least in my country, Holland) graded mainly by their so called output, ie. publications in scientific journals. How likely is it for such an electronic journal to be appreciated on this level. More precisely, will there be a change that it will be considered for citation indexes and the like?

From: **George Greenwade**
Date: 29 Sep 1993

Eric van Ammers posted: I like the ida very much, but my mainproblem is the following. Scientists in a university environment are (at least in my country, Holland) graded mainly by there so called output, ie. publications in scientific journals.

I think your "output" model can be expanded to most academics in most fields in most nations (so long as the periodical is recognized as being within the generalized field of the individual). The concept of electronic publishing is new to the universe, so how it is perceived is undoubtedly still subject to question. Should a new untenured faculty member seeking recognition publish in this media if alternate already-recognized media exists? If the output model is followed as it presently stands, very probably not; (s)he'd be much beter off following the traditional course. More to the specific point of what I asked (and everyone who has responded about this -- thanks, you've not been forgotted; I've simply been swamped in recent days, but you'll hear):

How likely is it for such an electronic journal to be appreciated on this level. More precisely, will there be a change that it will be considered for citation indexes and the like?

This is a semi-unique case, IMO (or following Joachim, it should really be IMNSHO, which is basically how you can always read my IMO's) as the area of literate programming is not presently being served by the traditional media, at least insofar as I can discern. If literate programming (or any underserved field) is indeed an area of research and discourse, an electronic journal may be one of the singly most appreciated periodicals within its field -- and very likely will show up in citation indices (I know of two instances presently in psychology where electronic periodicals are routinely included in citation indices).

Is this a risky venture for the author? Yes, as there are few existing models to compare it against. Is it a venture with potentially high merit? Yes, as it facilitates discourse in an underserved area. Is this a project which has a multitude of spin-offs? Yes, as it assists in developing the model which subsequent tasks in other areas may compare themselves to. Is this an "open" project? Yes, due to an entirely different set of costs, academics, practitioners, and developers may have complete exchanges since the cost and page limitations are removed. Is this a grounds for extending the field covered by the periodical? Yes, due to quick turnaround, publishing

queues are minimized -- meaning that research, developments, and ideas can quickly come into the focus of discussions (I don't know about your field(s), but in economics, the publishing queues for some journals now approach 3 years, meaning that the articles are stale by the time readers actually get to them).

In other words, can I (or anyone) promise that publishing in such a forum is a guarantee for parallel recognition to publishing in, say, the ACM? No. However, can you (or anyone) promise that publishing in such a forum will not be viewed as superior to the ACM within a short period of time if the contents of the effort are on-point and extensible? No.

From: **Lee Wittenberg**
Date: 29 Sep 1993

Eric van Ammers writes: I like the idea very much, but my mainproblem is the following. Scientists in a university environment are (at least in my country, Holland) graded mainly by there so called output, ie. publications in scientific journals. How likely is it for such an electronic journal to be appreciated on this level. More precisely, will there be a change that it will be considered for citation indexes and the like?

We have the same problem here in the States. However, if we think the electronic journal is a Good Thing (and I do), I say we should ``damn the torpedoes; full speed ahead!'' The Internet is changing the way Academia works, and it is up to us to educate our administrators as to the way things work today. This discussion group, for example, has been worth more to me than a dozen traditional conferences or journals would have been. The only downside is not being able to get together over coffee or lunch and talk things over face to face, although in some cases that might be an advantage :-).

From: **Marcus Speh**
Date: 29 Sep 1993

Marcus: We would like to bundle any sort of "virtual" educational activity -- a journal which is unlikely to ever appear on real paper fits in this category. We already have: a list of electronic consultants, a collaborative (Hyper-) textbook project, an internet meta library, and a virtual campus.

Eric van Ammers: I like the idea very much, but my main problem is the following. Scientists in a university environment are (at least in my country,Holland) graded mainly by there so called output, ie. publications in scientific journals. How likely is it for such an electronic journal to be appreciated on this level. More precisely, will there be a change that it will be considered for citation indexes and the like?

It is our long-term goal to create an fully accredited online university. GNA is also the world's first virtual corporation, and therefore also serves as a testbed for running organizations in the 21st century. That any kind of "virtual" activity still carries the stigma of non-seriousness is mainly due to the fact that the whole field is very young and under hectic development - count the articles on the "Information super highway" in international news magazines...I think the world doesn't have a definition for "multimedia" yet -- it is important to start a non-profit enterprise right now.

Nobody can foresee how accepted such a (purely electronic) journal will be. In High Energy Physics I have witnessed the rise of preprint bulletin boards which are in fact so much wanted by people (working scientists, between Bombay and Berkeley) that they endanger the existence of regular journals with their paper and staff costs etc. Our central library (of the biggest HEP lab in Germany) has almost ceased to distribute anything else but electronic preprints--I take things like this as a sign for change. This is not to say that mechanisms like peer review etc would (or should) be given up--their adequate translations for electronic media have to be found (examples do exist). Also I second Norman's view that "big names" are needed to convince people of the nobility of the endeavour.

From: **Stephen Fulling**
Date: 29 Sep 1993

What would the group think of a "refereed electronic periodical" for literate programming[?]

In principle I am very interested in this. However, there are some practical problems that need to be considered. First, in what format will the journal be distributed? Considering the subject, surely it would be a frustrating travesty to be limited to a pure ASCII end product. So, do we get LaTeX files? PostScript? Example webs in their original source code? (If so, I need to have every tool -- AWEB, BWEB, CWEB, ... -- installed on my workstation, just to read the journal.) All of the above?

It will be a long time before any major print periodical develops ...

Well, if the whole thing gets into PostScript, or even TeX, form, then it will BE a print journal for anybody who chooses to send it through a laser printer. More seriously, if we expect that most readers will be saving trees and reading the journal on a screen previewer, then consideration should be given to making the page shape landscape rather than portrait. (I try to do this myself with preprints I receive electronically, by halving the \vsize, but this often plays havoc with the author's carefully adjusted page layout.)

Second, will the editor(s) perform the usual editorial functions of copy editors of print journals, including correcting typographical and grammatical errors and beating everything into a reasonably uniform format? I think this is essential for respectability, for being taken seriously as a "real" journal of high standards. Furthermore, this task is even more important (and difficult) if we have the technological

problem of making TeX files compilable on every subscriber's system (see above).

But this task (or even the simpler one of sending files back and forth between authors and referees) is horrendous -- a full-time job. At a minimum I think it will require a secretarial-level person full time and a professional-level person half time. So now we need a source of salary money.... Help!

From: **Stefan Farestam**
Date: 29 Sep 1993

Eric van Ammers posted: I like the idea very much, but my mainproblem is the following. Scientists in a university environment are (at least in my country, Holland) graded mainly by there so called output, ie. publications in scientific journals. How likely is it for such an electronic journal to be appreciated on this level. More precisely, will there be a change that it will be considered for citation indexes and the like?

Just to fuel this debate further (in a positive direction), I'm including below two announcements of electronic journals within the field of numerical analysis. If someone pursues this idea further, which I definitely think should be done, then it might be worth contacting the editors of those journals to maybe gain some valuable insights. Note that both of the journals listed below will also be printed in a small quantity, so as to be available by inter library loans.

Call for Papers

Electronic Transactions on Numerical Analysis

Scope:

Electronic Transactions on Numerical Analysis (ETNA) is an electronic journal for the publication of significant new and important developments in numerical analysis and scientific computing. Papers of the highest quality that deal with the analysis of algorithms for the solution of continuous models and numerical linear algebra are appropriate for ETNA, as are papers of similar quality that discuss implementation and performance of such algorithms. New algorithms for current or new computer architectures are appropriate provided that they are numerically sound. However, the focus of the publication should be on the algorithm rather than on the architecture. The journal is published by the Kent State University Library in conjunction with the Institute of Computational Mathematics at Kent State University. Mathematical Reviews will receive all papers accepted for publication in the journal and review them as appropriate. ETNA is registered with the Library of Congress and has ISSN 1068-9613.

Dissemination:

On a quarterly basis, accepted manuscripts will be posted in a directory which is publicly accessible through Internet. The titles and abstract of these manuscripts will be e-mailed to registered departments and individuals and posted on public bulletin boards such as NA-digest. An individual who wishes to obtain a copy of a current or back manuscript can get a copy through anonymous FTP or by using a netlib-type mailer. We also plan to install Gopher. All manuscripts will be available in Post Script format. The first issue of ETNA will appear September 1, 1993. Funds made available by the Kent State University Library and the Kent State University make free subscription possible for at least three years. After this time period we may have to charge an annual fee from institutional subscribers. Since the operating costs for the journal are low, we envision that this fee will not be above $100 for institutional subscribers. Everybody at the subscribing institution will have access to ETNA by FTP, a netlib-type mailer or Gopher. In addition, articles in ETNA can be obtained through interlibrary loan from Kent State University Library.

To register to receive ETNA's quarterly titles and abstract lists, please send an e-mail message to etna@mcs.kent.edu. The subject of the message should be: ETNA registration. Titles and abstracts of papers published in ETNA will be e-mailed quarterly to the return addresses of all such requests. Inquiries for further information should also be e-mailed to etna@mcs.kent.edu.

Submission, Acceptance and Refereeing:

Authors will normally submit papers for publication via e-mail, and they will be required to submit their manuscript in LaTeX or TeX using macros we provide. Requests for macros can be sent by e-mail to etna@mcs.kent.edu. All papers will be refereed. As soon as a paper has been accepted for publication in ETNA, it will be entered into the ETNA data base. There are no annual page limitations, and, therefore, we are in a position to publish accepted manuscripts faster than many other journal. Manuscripts can be submitted NOW by sending them to the address etna@mcs.kent.edu.

Current Editorial Board:

```
L. Reichel          Kent State University
editor-in-chief     reichel@mcs.kent.edu

R.S. Varga          Kent State University
editor-in-chief     varga@mcs.kent.edu

A. Ruttan           Kent State University
managing editor     ruttan@mcs.kent.edu
```

```
G.S. Ammar          Northern Illinois University
J.W. Demmel         University of California, Berkeley
J.J. Dongarra       University of Tennessee
I.S. Duff           Rutherford Appleton Laboratory
M. Eiermann         University of Karlsruhe
J.A. George         University of Waterloo
G.H. Golub          Stanford University
W.B. Gragg          Naval Postgraduate School
M.H. Gutknecht      Swiss Federal Institute of Technology
V. Mehrmann         Technical University of Chemnitz-Zwickau
D.C. Sorensen       Rice University
G.W. Stewart        University of Maryland
O.B. Widlund        New York University
```

ELECTRONIC JOURNAL OF DIFFERENTIAL EQUATIONS. (EJDE)

The EJDE is published by Southwest Texas State University and the University of North Texas. This is a strictly electronic journal: Articles are to be submitted and then provided to the mathematical community electronically.

SCOPE

The EJDE will accept only first-rate original work, subject to as rigid a peer review process as is applied by the finest of today's journals.

DISSEMINATION

Abstracts of articles will be sent to subscribers as soon as accepted for publication (free of charge). Manuscripts and abstracts will be posted in a directory which is publicly accessible through Internet. Also, the American Mathematical Society will provide access to this journal through the E-math gopher.

Manuscripts will be available as TeX files. Which means that your local computer needs TeX processing facilities. Manuscripts can be obtained also as also as DVI or POSTSCRIPT files. Hard copies will be preserved for posterity. The publishers will originate and maintain copies at the libraries of both institutions. Photocopies of articles can be obtained from these libraries using the Interlibrary Loan system. (We are in the process of obtaining the ISSN number.)

For information through internet make sure your computer emulates a VT100 terminal and the type "telnet ejde.math.unt.edu" at the login prompt type "gopher" (without quotations), then select 1 for EJDE and follow the instructions on the screen.

EDITORIAL BOARD

```
P. Bates (Brigham Young University)
A. Bloch (Ohio State University)
J. Bona (Pennsylvania State University)
L. Caffarelli (Institute for Advanced Study)
C. Castillo-Chavez (Cornell University)
C. Chui (Texas A & M University)
M. Crandall (University of California at Santa Barbara)
E. Di Benedetto (Northwestern University)
G. B. Ermentrout (University of Pittsburgh)
J. Escobar (Indiana University)
L. C. Evans (University of California at Berkeley)
J. Goldstein (Louisiana State University)
C. Groetsch (University of Cincinnati)
I. Herbst (University of Virginia)
C. Kenig (University of Chicago)
R. Kohn (Courant Institute)
A. Lazer (Miami University)
J. Neuberger (University of North Texas)
P. Rabinowitz (University of Wisconsin)
R. Shivaji (Mississippi State University)
R. Showalter (University of Texas)
H. Smith (Arizona State University)
P. Souganidis (University of Wisconsin)
N. Walkington (Carnegie-Mellon University)
P. Waltman (Emory University)
```

SUBMISSIONS

Submissions should be files in one of the following formats: TeX, LaTeX, AMS (LaTeX or TeX). Graphics can be attached using either

PicTeX or Postcript. There is no page charge. We are accepting manuscripts NOW; send your files via E-mail to "editor@ejde.math.unt.edu". Please keep a copy of your submissions; we are not responsible for lost files.

COPYRIGHTS

By submitting a manuscript the author(s) agree that the copyright of the article is transferred to the publisher if and when the article is accepted for publication. Thanks for your attention and I am looking forward to see your sumissions to the EJDE.

Julio G. DiX
Department of Mathematics
Southwest Texas State University.

Notation issues

From: **Lee Wittenberg**
Date: 29 Sep 1993

A number of people have expressed opinions on the subject of literate programming and mathematical notation, and I guess I just can't keep my big mouth shut. It has often been noted that the bulk of programming language discussions center around points of syntax, rather than more substantive issues. While I agree that notation (language) shapes the way we think, and is therefore important, I believe that is important not to forget that the purpose of notation (language) is ~communication~. Therefore, I think that some kind of standardization of notation is useful, regardless of which notation is chosen as the standard.

Today's common programming languages provide multiple examples of what I mean. On the plus side, almost every language uses `*' for multiplication, and the rules for identifiers are pretty much the same. On the other hand, the assignment and logical operators vary widely (e.g., `=', `:=', `SET/TO' for the former).

The primary reason I prefer the "formatting" literate programming systems (like CWEB) to the "nonformatting" systems (like noweb) is that they provide a more standardized notation. I can look at a WEB, CWEB, or FWEB, and not have to remember what the assignment operator is for the particular base language -- they all use (or can use) `\gets'. Although I am (like many of you are) more comfortable with programming than mathematical notation, like Gulliver, I believe that getting the egg open is more important than which end you crack. Mathematical notation for logical operators has been around a great deal longer than any of the programming notations, so I'm perfectly willing to "standardize" on it, rather than search for the elusive perfect notation.

Literary nomenclature

From: **Joachim Schrod**
Date: 18 Oct 1993

Bart Childs writes: I also note that he called these small parts sections or modules. (He used the words interchangeably.) I think that is a far better term than scraps because they were intentional, not leftovers as in most of the definitions of scrap.

I prefer the simple term `program part'. For me, section is ok most of the time, but sometimes one must stress the difference between this `numbered entities' and their subparts. I _hate_ the term module. (In fact, Klaus and myself were one of the pushers to delete this word of CWEB.) A module is an entity with a distinct specification and implementation. One does not find this distinction in WEB. The concept of a module, first outlined by Parnas, later refined by Dennis, Ehrig & Mahr, and nowadays modernized by Booch, is a central one to all software engineering activities. It has _nothing_ to do with these WEB pieces. Please, don't throw away the CS terms we have worked so long for! For what it's worth, here is a definition of terms I use usually:

```
Before we start with an overview of the implementation I want to
explain the \cweb{} vocabulary I use while I guide you through this
document. The commonly used terms sometimes denote two entities, but
for the purpose of this style we need exact terms. I've tried to stick
to a "canonical" computer science terminology.

\begin{quotation}

I distinguish two different structures in a \cweb{} file: The {\sl
document structure\/} and the {\sl program structure}.

A \cweb{} document consists of a series of {\sl sections}. Within
this series some sections are especially emphasized, we call them the
```

```
{\sl main sections}. (They are also called {\sl starred sections},
since their corresponding \cweb{} tag is~|@*|.) These main sections
have a title, ordinary sections are untitled. A table of contents may
therefore list only the main sections. Note that there is no
hierarchy in the sections, they are all on the same level, ie, they
are numbered subsequently.

Each section consists of three parts: (1)~the {\sl documentation
part}, (2)~the {\sl definition part}, and (3)~the {\sl program part}.
Each of these parts can be empty. The documentation part is mostly text
with \LaTeX{} tags. In this text material from {\sl restricted program
mode\/} can appear. The definition part consists of a series of either
{\sl macro\/} or {\sl format definitions}. The program part is one
piece of a refinement, identified by a name (see below).

A \cweb{} program consists of a tree of {\sl refinements}. A
refinement is a list of program parts with the same name, ordered in
appearence. The root of the tree is the refinement with the special
name~|@c|. The program text is defined by the DFS (ie, infix-order)
traversal of the tree.

\end{quotation}

\noindent The terminology outlined above is an overspecification for
the \LaTeX{} style we're implementing here---nevertheless, the
context of my explanation should be clear now.%
```

(That's a verbatim copy from the CWEB style.) Any critic? comments?

From: **Bart Childs**
Date: 18 Oct 1993

Thanks to Joachim for his posting that addressed an omission from my previous posting. He (and others) were eloquent some time ago about the use of the term module ... I had intended to reference that and point out that we adopted some of that for the latest rewrite on web-mode earlier in the year. We call the sections that begin with @* (in WEB, CWEB, and FWEB) by the name `chapter' and the others simply `section.' Thus we have keybindings like C-c gc `go to chapter' ,,,

---

From: **Preston Briggs**
Date: 19 Oct 1993

I agree with Joachim Schrod about "module" already being heavily overloaded in computer science. Unfortunately, I also feel that "section" has an adequate meaning, at least in the context of Latex. Therefore, I've tried to use "scrap" where possible. Of course, people will be able to find errors in my usage, but "scrap" is the goal, even if I fall down occasionally in practice. Yes, "scrap" has a perhaps unfortunate implication of leftover or remnant that might put off some people. However, it has the big advantage of not having any more precise CS meaning.

From: **Lee Wittenberg**
Date: 19 Oct 1993

Joachim Schrod mentioned the terribleness (is that a word?) of the term "module" as used in the original WEB literature. I agree with him as to the superiority of "section" in this regard. So does DEK, apparently, as this terminology has been adopted in the most recent versions of CWEB.

However, my hands down favorite in the terminology game is Norman Ramsey's use of "chunks" in noweb. The word chunk carries no "excess baggage," as to both module and section (the former in programming, the latter in word processing). It's unambiguous to speak of "code chunks" and "text chunks" and the terms are readily understandable. Chunk also has the advantage that it's only one syllable, and therefore, easier to say. I don't mean to force anyone to adopt a new terminology, or to start any new religious wars. I just wanted to make the "chunk" terminology a bit more widely known.

From: **Bruce Dubbs**
Date: 19 Oct 1993

Joachim Schrod wrote: I prefer the simple term `program part'. For me, section is ok most of the time, but sometimes one must stress the difference between this `numbered entities' and their subparts. I _hate_ the term module. (In fact, Klaus and myself were one of the pushers to delete this word of CWEB.) A module is an entity with a distinct specification and implementation. One does not find

this distinction in WEB. The concept of a module, first outlined by Parnas, later refined by Dennis, Ehrig & Mahr, and nowadays modernized by Booch, is a central one to all software engineering activities. It has _nothing_ to do with these WEB pieces. Please, don't throw away the CS terms we have worked so long for!

Personally, I like the term `block', although `chunk' is OK too. Actually, a block of code can be considered in direct correlation to the cognitive chunks in psychology.

From: **Marcus Brown**
Date: 19 Oct 1993

Lee Wittenberg wrote: However, my hands down favorite in the terminology game is Norman Ramsey's use of "chunks" in noweb. The word chunk carries no "excess baggage," ...

Hear! Hear! "chunks" is an excellent choice. As a matter of fact, "chunks" does carry a certain amount of background, but it is very appropriate in this case. The term "chunk" is often used in discussing how many items a person can keep in short-term memory at one time: the magic number 7 +/- 2 chunks, usually where each chunk is a single idea/concept/unit/... These chunks may be decomposed into smaller chunks, and will be different for different people. For example, an entire chessboard with pieces across it might be a single chunk to a chess master, while it might be several chunks for the rest of us. On the other hand, for an good programmer, a `push' or `pop' or even `bubble sort' might be a single chunk. Any of these chunks could be broken down into smaller chunks or steps later.

The meaning carries over nicely, because we can talk about a `chunk' of code as being a single unit, even if it is later broken down into further chunks. I really like the term `chunks,' and hereby go on record as recommending it for canonization as the _APPROVED_ term for the code fragments (with associated documentation, macros, etc.) which make up a WEB.

From: **Eric van Ammers**
Date: 20 Oct 1993

Lee Wittenberg writes: However, my hands down favorite in the terminology game is Norman Ramsey's use of "chunks" in noweb. The word chunk carries no "excess baggage," as to both module and section (the former in programming, the latter in word processing). It's unambiguous to speak of "code chunks" and "text chunks" and the terms are readily understandable. Chunk also has the advantage that it's only one syllable, and therefore, easier to say.

There is another, much more important reason to standarize on the use of the word "chunk". Psychologist use "chunk" to identify the mental units one manipulates in his/her wetware (brain) when we are thinking. In a literate program we try to document individually the chunks that play a role in understanding a program.

From: **Joachim Schrod**
Date: 20 Oct 1993

Marcus Brown writes: I really like the term `chunks,' and hereby go on record as recommending it for canonization as the _APPROVED_ term for the code fragments (with associated documentation, macros, etc.) which make up a WEB.

I like the term `chunks', too. But, not as a replacement for `section' -- as a replacement for `program part'. Or did I misunderstand you, and you talk about the same? Nevertheless, I would also like to keep the term `refinement' as a notion for the catenation of all chunks with the same name. I.e., for one node in the tree that constitutes the `real' source, after all. I think the disctinction is important, since we're on different abstraction levels here.

From: **Christine Detig**
Date: 21 Oct 1993

Bruce Dubbs writes: Personally, I like the term `block', although `chunk' is OK too. Actually, a block of code can be considered in direct correlation to the cognitive chunks in psychology.

In many programming languages, `block' is used for parts of code which have a common scope. Thus, our chunks would in general spread over several blocks or contain only parts of them, which seems to make the term unappropriate to me. I think `chunk' is fine, but as a non-native speaker, I'm not sure if this word contains some sloppyness --- I mean, the kind of words you say but don't write. Could some native speakers, probably from both sides of the Atlantic, comment on this? Good documentation, it seems, is like the weather: everyone is talking about it, but few people are doing anything about it.

From: **Colman Reilly**
Date: 21 Oct 1993

Christine Detig writes: I think `chunk' is fine, but as a non-native speaker, I'm not sure if this word contains some sloppyness --- I mean, the kind of words you say but don't write. Could some native speakers, probably from both sides of the Atlantic, comment on

this?

Chunk is an execellent word to use actually, as it doesn't have any other technical meaning I'm aware of. As someone new to literate programming I found the word scrap very strange in this context - my code isn't a _scrap_ or leftover anything, no matter what my lecturers say. :-)

From: **Marcus Brown**
Date: 21 Oct 1993

Joachim Schrod wrote: I like the term `chunks', too. But, not as a replacement for `section' -- as a replacement for `program part'. Or did I misunderstand you, and you talk about the same?

I would like to use the term in referring to both the program part and the associated documentation. After all, in a 'Literate' program, the code is not complete without the accompanying documentation, etc. However, I would not object to the term being used specifically with reference to the code alone.

From: **Michael Koopman**
Date: 21 Oct 1993

Marcus Brown wrote: I would like to use the term in referring to both the program part and the associated documentation. After all, in a 'Literate' program, the code is not complete without the accompanying documentation, etc. However, I would not object to the term being used specifically with reference to the code alone.

How about chunklet for the code or documentation part. Cryptic code or poor documentation could be "dark" chunklet and code that took forever to write could be "milk" chunklet.

From: **Lee Wittenberg**
Date: 21 Oct 1993

Joachim Schrod writes: I like the term `chunks', too. But, not as a replacement for `section' -- as a replacement for `program part'. Or did I misunderstand you, and you talk about the same?

Generally, I use `chunk' for program part, but the noweb documentation refers to "text chunks" and "code chunks," and I use those terms as well.

Nevertheless, I would also like to keep the term `refinement' as a notion for the catenation of all chunks with the same name. I.e., for one node in the tree that constitutes the `real' source, after all. I think the disctinction is important, since we're on different abstraction levels here.

That's a good point. I like the term `refinement' in this context. It works well with the way `refinement' is used in the ABC programming language.

From: **C. M. Sperberg-McQueen**
Date: 22 Oct 1993

I think `chunk' is fine, but as a non-native speaker, I'm not sure if this word contains some sloppyness --- I mean, the kind of words you say but don't write. Could some native speakers, probably from both sides of the Atlantic, comment on this?

I think your instinct is correct, Christine: at least to this native speaker of AmE, 'chunk' is marked for informality, and is more likely to be found in spoken than in formal written English. That is, of course, part of its appeal to some of us (me, I like 'scrap' even better, partly because it is an even less predictable term and partly because in the context I work in, 'chunk' has already been assigned a technical meaning as the name of a class of SGML elements). So it is not at all out of place in normal AmE computer science-type discourse, but it is perhaps the kind of word (like 'memory' and 'flavor') bound to meet with the disapproval of Edsger Dijkstra and others who share some of his less flexible notions concerning proper terminology.

From: **Lee Wittenberg**
Date: 22 Oct 1993

Christine Detig writes: I think `chunk' is fine, but as a non-native speaker, I'm not sure if this word contains some sloppyness --- I mean, the kind of words you say but don't write. Could some native speakers, probably from both sides of the Atlantic, comment on this?

As a native speaker (West Side of Atlantic/East Side of Pacific chapter -- we shouldn't forget the other ocean; native English speakers seem to like distancing themselves from each other :-), `chunk' has the right ring to my ears. However, having been born in Brooklyn, I might have to be disqualified. `Chunk' also works well on paper, having acquired legitimacy from Psychology (as a number of

participants in this discussion have already pointed out).

From: **Bruce Dubbs**
Date: 22 Oct 1993

Lee Wittenberg wrote: As a native speaker (West Side of Atlantic/East Side of Pacific chapter -- we shouldn't forget the other ocean; native English speakers seem to like distancing themselves from each other :-), `chunk' has the right ring to my ears. However, having been born in Brooklyn, I might have to be disqualified. `Chunk' also works well on paper, having acquired legitimacy from Psychology (as a number of participants in this discussion have already pointed out).

The problem with `chunk' is that it does have this use in Psychology. The more complete term is `cognitive chunk'. I have proposed the term `cognitve block' in a paper now being refereed. In writing code, using the style of literate programming or not, the author presents code in groupings. The degree of mapping of these groupings or `cognitive blocks' to the reader's psychological `cognitive chunks' really dictates the readability or understandability of the code. My research indicates that WEB style literate programming tends to provide a better mapping between blocks and chunks.

From: **Kasper OEsterbye**
Date: 25 Oct 1993

In this wonderful line of chunks and scraps, I would as a native speaker (of Danish) and a naive speaker (of english) just throw in that I use the word "fragment". I kind of like it because it is exactly what it should be in danish, and seems to work just fine in foreign too. Also it sounds more formal and better suited for papers.

From: **Eric van Ammers**
Date: 25 Oct 1993

I'm surprised that this discussion about 'chunks' has not referred to serious research that has been done already. For instance Soloway [1,2] finds that programmers tend to think in 'plans' and it looks like these plans are closely related to the refinements we want to document in our literate programs. 1. Soloway E., Learning to Program= Learning to Construct Mechanisms and Explanations, CACM 29, 9, 850-858 (1986) 2. Soloway E., Designing Documentaion to Compensate for Delocalized Plans, CACM 31, 11, 1259-1267 (1988)

From: **Mike Yoder**
Date: 25 Oct 1993

Let me, as one whose native language is English, loudly endorse Kasper OEsterbyte's use of "fragment." It has the correct meaning in plain, non-technical English, and doesn't collide with existing jargon.

From: **Lee Wittenberg**
Date: 25 Oct 1993

Kasper Osterbye suggests: In this wonderful line of chunks and scraps, I would as a native speaker (of Danish) and a naive speaker (of english) just throw in that I use the word "fragment". I kind of like it because it is exactly what it should be in danish, and seems to work just fine in foreign too. Also it sounds more formal and better suited for papers.

`Fragment' is certainly acceptable, but I think the final sentence above is an argument ~against~ it. Part of the problem with formal papers is that they use too many 2-bit words. As I keep telling my students: "People who use long words either don't know what they're talking about, or don't want you to know what they're talking about." My vote still goes to `chunk,' the only one-syllable contender.

From: **Michael Koopman**
Date: 25 Oct 1993

Mike Yoder wrote: Let me, as one whose native language is English, loudly endorse Kasper OEsterbyte's use of "fragment." It has the correct meaning in plain, non-technical English, and doesn't collide with existing jargon.

There is technical jargon which uses the word `fragment'. Aside from sentence fragment. The word `fragment' is used to describe the pieces scattered about after a shell or other item is fractured, such as by an explosive charge. The weaving and tangling does "anti-fracture" the bits of text and code scattered about the web. I still like the taste of `chunklet' better than `fragment,' however.

From: **Lewis Perin**
Date: 25 Oct 1993

Lee Wittenberg writes: My vote still goes to `chunk,' the only one-syllable contender.

My heart still belongs to the saltier `scrap', equally monosyllabic and properly evocative of a piece of paper that can be cut and pasted. (Of course, it also evokes the dog's dinner, but I happen to *like* that association.)

From: **Christian Lynbech**
Date: 26 Oct 1993

Let me, as one whose native language is English, loudly endorse Kasper OEsterbyte's use of "fragment." It has the correct meaning in plain, non-technical English, and doesn't collide with existing jargon.

In fact, the term is used (and probably in more instances than the following). A locally developed OO programming language, the Beta language, uses a `fragment system' as the main vehicle of tying program modules together. An interesting fact is that this fragment system is roughly as powerfull as the tangle part of (say) nuweb, i.e. you define named fragments which are then inserted into `slots'. The system is, as I understand it, derived from grammars (in some appropriate sense), with slots corresponding to non-terminals.

Whether this counts against or for `fragment', is a good question. I have until this thread started used the term `scrap', which I got from spider webs langauge defition files. The process of prettyprinting the code is described as forming bigger scraps from smaller scraps, and though this has nothing to do directly with tangling, it corresponds nicely to my understanding of the tangling process. And since I also find chunk overly informal, I will support the scrap camp.

From: **Chris Gray**
Date: 26 Oct 1993

Bruce Dubbs writes: The problem with `chunk' is that it does have this use in Psychology. The more complete term is `cognitive chunk'. I have proposed the term `cognitve block' in a paper now being refereed.

Ooerr. In that context `cognitive block' makes me think of `mental block' or `writer's block'.

From: **Bruce Dubbs**
Date: 29 Oct 1993

Bruce Dubbs writes: The problem with `chunk' is that it does have this use in Psychology. The more complete term is `cognitive chunk'. I have proposed the term `cognitve block' in a paper now being refereed.

Chris Gray writes: Ooerr. In that context `cognitive block' makes me think of `mental block' or `writer's block'.

I see. We are talking about nouns and you think about verbs. ;-)

Results of survey on prettyprinting

From: **Conrado Martinez-Parra**
Date: 20 Oct 1993

In my opinion, prettyprinting can be a useful feature but it is not necessary to provide it as a built-in feature of a WEB system. What is really needed is that the WEB system produces intermediate files, with standard 'formats'. Different specialized tools can then work on the appropriate file to generate xrefs, indexes, prettyprinted output, etc. This is the kind of approach that Norman Ramsey used in his noweb system, and it is the right one, in my opinion. If one likes prettyprinting, the only thing that is needed is a program that takes the weave'd file and includes prettyprinting macros in the code sections. If you don't like prettyprinting, don't use the prettyprinting program.

I will use a particular example to make clear what I mean. Suppose we are working with noweb and LaTeX. After weaving the web file (.nw), one obtains a LaTeX file (.tex) where the code goes inside 'code' environments, for example,

```
\begin{code}{}
...
int gcd(int x, int y)
{
  while (x != y)
       ...
}
\end{code}
```

This 'code' environment is very similar to the standard 'verbatim' environment and therefore you'll not have prettyprinting output. The prettyprinting tool takes this LaTeX file as input, leaves everything outside the 'code' environments untouched and includes the prettyprinting macros within the code sections:

```
\begin{ppcode}{}
...
|int|\ |gcd|(|int| x, |int| y)
\{
\while (x \ne y)
...
\end{ppcode}
```

I have developed such a 'ppcode' environment and the prettyprinting program for Dijkstra's command language. The pipeline 'noweb | d2tex' works rather well (although, it is a beta version and the prettyprinting output sometimes looks terrible :-)). Finally, it seems to me that a tool for generating such prettyprinters would be very useful and not very difficult to produce (the Spidery WEB system includes, in some sense, such a generator, doesn't it?).

From: **Norman Ramsey**
Date: 25 Oct 1993

Thanks to the 17 respondents to my survey. Here are the results. Recall the scale:

```
>   -2 not useful
>   -1 seldom useful
>    0 often useful
>   +1 useful; can get along without, but only painfully
>   +2 indispensible
>   NA I never heard of this, have no opinion
```

In addition, I asked what people preferred among several alternatives. Some people gave preferences; others preferred to rate the alternatives.

```
prettyprinting (PP)          rated -0.3 +/- 0.3 (17 ratings)
table of contents (TOC)      rated +1.4 +/- 0.2 (17 ratings)
identifier cross-ref (IX)    rated +0.6 +/- 0.2 (16 ratings), 1 NA
  automatically (AUTOIX)     rated +1.0 +/- 0.4 (4 ratings), preferred 8 times
  by the programmer (USERIX) rated -0.8 +/- 0.5 (4 ratings), preferred 1 time
  semi-automatic (SEMIIX)    rated -0.7 +/- 0.3 (3 ratings), preferred 4 times,
                                                  OK 3 times, one ``?''
index of identifiers (II)    rated +1.3 +/- 0.2 (16 ratings)
  automatically (AUTOII)     rated +1.5 +/- 0.3 (4 ratings), preferred 8 times,
                                                 one ``?''
  by the programmer (USERII) rated -0.3 +/- 0.9 (4 ratings), preferred 1 time
  semi-automatic (SEMIII)    rated -0.7 +/- 0.3 (3 ratings), preferred 3 times,
                                                 OK 3 times, one ``?''
chunk cross-reference (CX)   rated +1.8 +/- 0.1 (17 ratings)
chunk index (CI)             rated +1.0 +/- 0.2 (17 ratings)
chunk numbers (CN)           rated +0.7 +/- 0.3 (14 ratings), preferred 1 time
numbers for cross-reference
  page numbers (PAGENO)      rated +0.5 +/- 0.3 (6 ratings), preferred 2 times
  consecutive (CONSEC)       rated +0.4 +/- 0.5 (5 ratings), preferred 3 times
  `1, 2a, 2b, &c' (SUBPAGE)  rated +1.3 +/- 0.3 (4 ratings), preferred 8 times
```

Two respondents mentioned that they wanted fully automatic indexing but that the ability to supplement the index manually was important. Here are the ratings in order of importance. I've left out some I took to be preferences. As you can see, differences between nearby features are not significant. Prettyprinting comes out dead last and has the largest margin of difference between it and its neighbors.

```
chunk cross-reference CX         rated +1.8 +/- 0.1 (17 ratings)
automatic indexing    AUTOII     rated +1.5 +/- 0.3 (4 ratings)
table of contents     TOC        rated +1.4 +/- 0.2 (17 ratings)
index of identifiers  II         rated +1.3 +/- 0.2 (16 ratings)
sub-page numbers      SUBPAGE    rated +1.3 +/- 0.3 (4 ratings)
chunk index           CI         rated +1.0 +/- 0.2 (17 ratings)
chunk numbers         CN         rated +0.7 +/- 0.3 (14 ratings)
```

```
identifier cross-ref  IX         rated +0.6 +/- 0.2 (16 ratings)
page numbers          PAGENO     rated +0.5 +/- 0.3 (6 ratings)
consecutive chunk nos CONSEC     rated +0.4 +/- 0.5 (5 ratings)
prettyprinting        PP         rated -0.3 +/- 0.3 (17 ratings)

Finally, here is the original survey and the coding of responses.

Please rate these features:
  prettyprinting
  table of contents
  identifier cross-reference (local, on-page info about local defns & uses)
    if useful, do you prefer that definitions and uses be identified
      -- completely automatically
      -- by the programmer
      -- programmer marks definitions, uses are identified automatically
  index of identifiers at end of document
    if useful, do you prefer that definitions and uses be identified
      -- completely automatically
      -- by the programmer
      -- programmer marks definitions, uses are identified automatically
  module/chunk cross-reference (automatically generated)
  module/chunk index (automatically generated)
  module/chunk numbers (are they important, or do page numbers suffice?)
  if you use cross-reference or index info, what numbers should be used?
    -- use page numbers
    -- use (consecutive) module/chunk numbers
    -- use module/chunk numbers, but number them by page of appearance
       (e.g. 1, 2a, 2b, 3, 4a, 4b, 4c) instead of consecutively.

Answers: (X means preferred, OK means not preferred but acceptable)

(Name of respondent deleted)
 PP=-1 TOC=+1 IX=+1 AUTOIX=+1 USERIX=-1 SEMIIX=0 II=+1 AUTOII=+1
 USERII=-1 SEMIII=0 CX=+1 CI=+1 CN=0 PAGENO=+1 CONSEC=0 SUBPAGE=+1
(Name of respondent deleted)
 PP=-1 TOC=+2 IX=0 SEMIIX=X II=+2 SEMIII=X CX=+2 CI=+1 CN=0 SUBPAGE=X
(Name of respondent deleted)
 PP=0 TOC=+2 IX=+2 AUTOIX=+2 USERIX=-2 SEMIIX=-1 II=+1 AUTOII=+2
 USERII=-2 SEMIII=-1 CX=+2 CI=0 CN=0 PAGENO=0 CONSEC=+1 SUBPAGE=+2
(Name of respondent deleted)
 PP=0 TOC=+2 IX=+2 AUTOIX=X SEMIIX=OK AUTOII=X SEMIII=OK CX=+2 CI=+2
 SUBPAGE=X
(Name of respondent deleted)
 PP=-1 TOC=+1 IX=+2 AUTOIX=+1 USERIX=0 SEMIIX=-1 II=+2 AUTOII=+1
 USERII=0 SEMIII=-1 CX=+2 CI=+2 CN=0 PAGENO=0 CONSEC=0 SUBPAGE=+1
(Name of respondent deleted)
 PP=+2 TOC=+2 IX=+1 AUTOIX=0 USERIX=0 SEMIIX=? II=+2 AUTOII=+2
 USERII=+2 SEMIII=? CX=+2 CI=+2 CN=+2 PAGENO=+2 CONSEC=+2 SUBPAGE=X
 AUTOII=?
(Name of respondent deleted)
 PP=-2 TOC=0 IX=0 AUTOIX=X II=0 AUTOII=X CX=0 CI=0 PAGENO=0
(Name of respondent deleted)
 PP=-1 TOC=0 IX=0 AUTOIX=X II=+1 AUTOII=X CX=+1 CI=0 CN=X
 CONSEC=X SUBPAGE=X
(Name of respondent deleted)
 PP=+1 TOC=+1 IX=-1 AUTOIX=X II=+1 AUTOII=X CX=+2 CI=+1 CN=+2 CONSEC=X
(Name of respondent deleted)
 PP=+1 TOC=+2 IX=0 AUTOIX=X SEMIIX=OK II=+1 AUTOII=X
 SEMIII=OK CX=+2 CI=+2 CN=+2 SUBPAGE=X
(Name of respondent deleted)
 PP=0 TOC=0 IX=+1 AUTOIX=X  II=+1 AUTOII=X CX=+2 CI=+1 CN=0 SUBPAGE=X
(Name of respondent deleted)
 PP=+1 TOC=0 IX=NA AUTOIX=X II=+2 AUTOII=X CX=+2 CI=0 CN=+2 CONSEC=X
(Name of respondent deleted)
 PP=+1 TOC=+2 IX=0 SEMIIX=X II=+2 SEMIII=X CX=+2 CI=+2 CN=-2
 PAGENO=X
(Name of respondent deleted)
 PP=-1 TOC=+2 IX=0 AUTOIX=X SEMIIX=OK II=+2 AUTOII=X SEMIII=OK CX=+2 CI=+2
```

```
 CN=+1 SUBPAGE=X
(Name of respondent deleted)
 PP=-1 TOC=2 IX=0 USERIX=X II=2 USERII=X CX=2 CI=0 CN=0 PAGENO=X
(Name of respondent deleted)
 PP=-2 TOC=+2 IX=0 SEMIIX=X II=+2 SEMIII=X CX=+2 CI=+2 CN=+2 SUBPAGE=X
(Name of respondent deleted)
 PP=-1 TOC=+2 IX=+1 SEMIIX=X II=-1 CX=+2 CI=-1 CN=+1 PAGENO=0 CONSEC=-1
 SUBPAGE=+1
```

Some general thoughts on literate programming

From: **Venkataraman Chandrasekhar**
Date: 25 Oct 1993

In my nineteen years of working as a programmer (a field that I have considered leaving a couple of times because it is not satisfying enough), literate programming is one of the most exciting developments that I have come across. (The only other that even comes close, subjectively speaking of course, is Smalltalk.) I wouldn't have known about it if it weren't for the present newsgroup.

Some general thoughts. I had a chance to look at Norman Ramsey's paper (Software-Practice and Experience Vol.21(7),677-683) this weekend. He points out that 'The right model for a literate program is ... not the novel but the car repair manual.' In fact, terms like 'literate', 'publishable', 'books', etc. may cause companies (like ours) to 'run away' from these techniques (in the belief that these are for academia, and other types of companies). Yet, companies like ours can benefit a lot from these techniques. The advantage of these techniques (as I see it in my present state of understanding) is that programs written this way make it possible for a maintenance programmer to take effective action even when his level of familiarity is not high; car repair manual is a good analogy. For me, some of the best written manuals are the manuals that come with HP Laser Jet printers. Their purpose is to enable someone to act effectively (to solve a problem or achieve an intended result for instance, to change the toner cartridge) without going through a lot of preparation/reading/training. The manual makes up for the lack of preparation/reading/training. Techniques for communicating effectively in this way are available and it makes a lot of sense to make use of these in programming. 'literate' and the like have purposes such as 'amusing the reader', produce specific emotional effects,etc. These will be frowned upon as being inappropriate for serious business endeavors. Though, judicious use of these do make the process of using a manual/program quite entertaining and should indeed be welcomed. (The allusion to 'what a tangled web we weave' dawned on me this weekend. I also like the 'copyleft' policy of some GNU utilities.) As for emotional effects, having to make changes to a very badly written program, of course, has moved many to tears.

A final thought. B.F.Skinner has a theory of language (stated in Verbal Behavior, 1957). Programming behavior (including that done in a conventional language like English - which is documentation - and that done in a programming language which is of course code) can be profitably analyzed using Skinner's theories; I think that this a rich territory waiting to me mined. (I am aware that many people think of this approach as a 'dead' one - Chomsky etc. After about nineteen years of involvment in this as a hobby, I don't think so). For instance, some project/software methodologies actually make a programmer's job a lot harder/unsatisfactory; behavior analytic explanations shed good light on why this is so.

Also, the following may be of some interest to readers of present newsgroup: Skinner has a short paper that helps one enjoy the writing process more and produce better results. (I didn't use it in writing the present posting. I didn't want to go to the extra trouble it takes. I use it usually when I am struggling with a particular piece of writing.) If anyone is interested, please contact me; I will provide the reference. I also appreciate very much this opportunity to exchange ideas with (what I think are) kindred souls. For the record, I also want to say (1) English is a non-native language to me (though I have writing aspirations) (2) apologies to those who feel that this is wasteful of their resources (limited e-mail allowance).

From: **Thorbjoern Andersen**
Date: 26 Oct 1993

Venkataraman Chandrasekhar writes: Also, the following may be of some interest to readers of present newsgroup: Skinner has a short paper that helps one enjoy the writing process more and produce better results. (I didn't use it in writing the present posting. I didn't want to go to the extra trouble it takes. I use it usually when I am struggling with a particular piece of writing.) If anyone is interested, please contact me; I will provide the reference.

I am about to write my first larger Lit programme, and I would appreciate reading Skinners paper before starting. I intend to do this as right as I can. I agree with your comment on the LaserJet documentation. I installed a LaserJet 4L yesterday for the first time, and it was a breeze thanks to the excellent manual. I will take heed of your comment regarding using this as inspiration for my own work. So if you would provide references for the Skinner paper, as well as a ftp-location for Norman Ramsey's paper I would appreciate it very much.

From: Michael Koopman
Date: 26 Oct 1993

Thorbjorn Andersen wrote: I agree with your comment on the LaserJet documentation. I installed a LaserJet 4L yesterday for the first time, and it was a breeze thanks to the excellent manual. I will take heed of your comment regarding using this as inspiration for my own work.

This document is a User's Manual, not a Technical Manual. I am not criticizing the document or denying it's value to the intended audience. However, I do not believe this document is a good template for a literate program of significant depth. Of course, diagrams and directions for every contingency plausible owing to correct and incorrect use, and use under failure conditions for a complex program would be nice, but seems a daunting task. Much discussion centers around the methods for simplifying the complex program implementation into `User Manual descriptions' with literate techniques. I do not feel such efforts are strictly "academic" or infeasible in the "Real World' of "Real Programmers."

The diversity of the User's Manual and the Technical Manual seems relevant to questions concerning literate techniques which support multiple expositions for different audiences. It seems that some readers want a Literate Program to fulfill the User's Manual purpose while others desire a document oriented toward maintenance and enhancement of the code and algorithms. A document (web) which serves both purposes (weave*s*), and contains the implementation (tangle) may be the ideal.

I am about to write my first larger Lit programme, and I would appreciate reading Skinners paper before starting.

Skinner is nothing but a black box to me.

From: Venkataraman Chandrasekhar
Date: 26 Oct 1993

The subject is HP Users's Manual as an example of something that provides good instructions to its intended readers so that they can act effectively in solving their problems

This document is a User's Manual, not a Technical Manual. I am not criticizing the document or denying it's value to the intended audience. However, I do not believe this document is a good template for a literate program of significant depth. Of course, diagrams and directions for every contingency plausible owing to correct and incorrect use, and use under failure conditions for a complex program would be nice, but seems a daunting task. Much discussion centers around the methods for simplifying the complex program implementation into `User Manual descriptions' with literate techniques. I do not feel such efforts are strictly "academic" or infeasible in the "Real World' of "Real Programmers."

*The diversity of the User's Manual and the Technical Manual seems relevant to questions concerning literate techniques which support multiple expositions for different audiences. It seems that some readers want a Literate Program to fulfill the User's Manual purpose while others desire a document oriented toward maintenance and enhancement of the code and algorithms. A document (web) which serves both purposes (weave*s*), and contains the implementation (tangle) may be the ideal.*

Some responses to Michael Koopman's recent posting quoted above: I did not intend to imply that the HP User's Manual is a 'good template for a literate program ...' nor did I have 'diagrams and directions for every contingency ... ' in mind. What I was thinking was these: the HP User's Manual does a terrific job in helping its intended audience do user type functions - changing toner cartridges, loading different fonts,etc. Information is organized in an easy to find way. The whole experience is generally pleasing. Not a lot of prior experience is required. Not a lot of 'look under another topic' type of stuff is required. The whole thing is geared to the reader taking effective action. I can imagine a programmer's manual doing a similarly terrific job in helping its intended audience do maintenance programmer type of functions. To give an example (based on a recent experience): in some system, the requirement is to do one type of checksum if the year invloved is 94 and do another type of checksum if the year invloved is > 94. An existing program had code calculating checksums for 94 and 95. For 96 etc., it did nothing. This error had to be fixed. The maintenance programmer function involved here is to change a test such that for any year >94, the previous 95 logic is executed. I can imagine a good version of this system which provides most (if not all) of the advantages I have listed above for the HP User's Manual. Namely, what is done for the various years (for checksumming) should be very easy to locate by someone with no experience with the system (for instance, someone with 'college level curiosity'). In other words, a new maintenance programmer should be able to realize pretty easily from the program that nothing is done for the years >95. It should also be easy to realize that the requirement for years >95 is the same as that for 95. (In fact, if the original programmer had organized her program/document this well, she would have realized that she is not providing any checksumming function for >95.) In sum, if the program/document makes it very easy for someone new to make this change and do it without getting irritated (just like using the HP User's Manual one can change toner cartridges without much irritation; without such a nice manual, a few cusswords and kicking of the printer are highly likely), it has met the criterion of aiding/enhancing effective action by a new reader/user. Outside of such maintenance programmer's manual, of course, enduser(nonprogrammer) manuals are needed for people to type in commands to perform checksum on actual parts etc. Instructions for these readers may/may not belong in a literate program. (Of course, these manuals can be written providing the above benefits,

etc.)

Also, judicious decisions should be made (based on intended audience and several other factors) in areas like how much spoonfeeding to make etc. If a lot users are likely to be so angry that they cannot even see the print on a book when they are likely to use the book (as when going to the printer manual when their printer suddenly stops working), perhaps all the possible 'spoonfeeding' (i.e., diagrams and directions for every contingency plausible owing to correct and incorrect use, and use under failure conditions, etc.) should be considered. I certainly do not expect the program/document for the above case to tell me, for instance, 'if year = 96, add code to handle this'. If the program/document guides me quickly to go the section invloved, no other info is needed (other than what the reqmt for years >95 is). If the instructions quickly take me to a certain depth and if I am assured that the instructions are accurate, asking a maintenance programmer to dig through the next several levels is not unreasonable. Having to reinvent the wheel evey single time is what I want to avoid. Also, in my view, maintenance programmers needs are less important than endusers needs, if there is a contention for resources. However, most of the times both constituencies can be served well without one having to sacrifice for the other; what is in short supply is the knowhow/desire to do it.

I hope that these type of postings are of some value. If you guys start complaining, I will have to take my thoughts to alt.professional.wrestling. Also, my friend Ron Peithman told me yesterday that the following is the full quote: 'What a tangled web we weave, when we pratice to deceive'. Some project manager types that I know are indeed literate programmers without realizing that they are: they do weave tangled webs; practice quite a bit of deception.

Ordering Dependencies and Scoping

From: **Zdenek Wagner**
Date: 18 Nov 1993

Chris Nadovich said: Thank you, everyone that pointed out that there were now SuperDuper versions of WEB that answered most of my criticisims. I am now attempting to collect and evaluate some of these. With luck I'll find ONE version that has most of my wish-list entries in the 3-4 hrs per month I have free to experiment with new tools---we don't get much play time where I work. The only unanswered criticism was the issue of scoping. Steve Avery made an attempt to answer it, which I find inadequate.

I don't go along with your notion of having locality of reference in a WEB. The sort of code you are proposing is working against the grain of what literate programming is about - ie. lots of words not few words :-). If you must have several "Initialize" statements, then they should have different names. This then allows you to easily distinguish between different sections in both the index, list of sections, and code itself.

Clearly Steve understands my complaint about the lack of scoping in web. I know this, because he tells me what I do now. Every time I try to merge a perfectly good piece of code with another---even if they are targeted for separate files---I have to worry about name collisions. The ability to abbreviate scrap names makes this worse. I like all my C files to have the same basic format. Why do I have to rename all the sections in the top-level template when I apply it to a new module. This is unneeded work and it's error prone. Although I'd like a flexible scoping system, basic file target scoping would help me a lot. Here's one way to do it:

Make @i attach an invisible source filename code on the begining of every scrap name in the file that is included. The bulk of tangle and weave processing would treat these names as unique, but they'd look the same after they are TeX'ed. Next, rewrite the index generation stuff in weave to make this source filename visible again. Lastly, provide an override mechanism on the @i and the @< to force global scoping, deleting the invisible filename reference. I think that should do the trick.

I am afraid that generally files input @i are not easily includable into any web. However, if they contain only functions and/or class implementations, I feel better to build a private library. I do it with my functions and classes and I am rewriting the old libraries into CWEB. Thus my programs are smaller and compile faster. Just as you put #include <stdio.h> I use #include <myclass.h> or something similar. Then I specify the library in the project so that the linker can find it. If I don't remember something, I know there is another volume of a woven file and I have to look there for explanation.

From: **Tommy McGuire**
Date: 18 Nov 1993

Michael Koopman wrote: Two recent posts related to name spaces, one on scope and one on dependencies, reek of creeping featurism which I desire as a creepy feature user. Norman Ramsey boasts of file scoping capability in noweb (which can be managed with make utilities?). Stuart Ferguson has trouble collecting up C typedefs from "top-down" chunks.

Stuart Ferguson wrote [Ordering Dependencies]: One problem that I am noticing is that of final ordering of elements in the output program text. Like most one pass compilers, C requires that objects be declared before they are used. This lends a decidedly "bottom-up" quality to raw C code. I am finding that my literate programs tend to be more "top-down," with declarations of higher level objects before the lower level components which they require. The problem is that if I use some generic refinement like <<Public Data Types >> to collect up my typedef's, for example, then the objects are declared in the wrong order for the compiler.

The concepts described are not exactly clear to me. I would not want chunks to "float" about anywhere. Rather than levitating chunks I would like to chain them down. I believe the ancillary information should be dependency information. An index tag could allow dependency.

Pardon me for being terribly dense today, but is this a non-problem? If type Foo depends on the definition of type Bar, shouldn't there be a scrap

```
<<Public Data Types>>=
  <Define Type Bar>
  <Define Type Foo>
```

just to make the dependency explicit? Far be it from me to resist the addition of features of extreme complexity and beauty, of course.

From: **Steve Avery**
Date: 19 Nov 1993

Chris Nadovich talks about scoping and include files. To be quite honest, my original idea with the arbitrarily deep sectioning was to have included files be included at the current level or next level down (never did work out which was better) in the hierarchy. In this way, the top level of the hierarchy could simply be a place where the files were explained and included - supposedly making it possible to have a whole suite of files easily made into one document. (It would be possible to denote where a section is from by having the filename in the margin for each section, and possibly in the list of sections as well.)

I talked (or rather emailed back and forward) with Lee Wittenberg about this, and he pointed out a few flaws, the least of which was the hassle with what to do with the TeX waffle (sorry, forgot the terminology) at the front of each web file. There are other hassles too, but I'm sure there is a way around them given enough thought and coding (it might mean that some new requirements, like the waffle go in a separate file, need be imposed on webs).

Anyways, if this idea were to be implemented, it might make scoping easier (okay, it probably would). I still don't like the idea, but that doesn't mean it shouldn't be implemented for those that do. Oh, for those that haven't guessed, the web I'm talking about is CWEB.

p.s. Chris - if you want it so bad, consider putting it in yourself, and then giving Silvio a copy for a look-see. It may well get into the next release. p.p.s. I'd be interested in knowing what people think about the idea of including files at a certain level in the hierarchy. It may also be able to be over-ridden with a switch on @i (like @i0).

From: **Eitan Gurari**
Date: 19 Nov 1993

Steve Avery writes: Chris Nadovich talks about scoping and include files. To be quite honest, my original idea with the arbitrarily deep sectioning was to have included files be included at the current level or next level down (never did work out which was better) in the hierarchy. In this way, the top level of the hierarchy could simply be a place where the files were explained and included - supposedly making it possible to have a whole suite of files easily made into one document...

Hierarchical scoping of code fragments seems to me to be the way to follow in two-dimensional (e.g., hypertext-oriented) environments that exhibit these relationships in a natural manner. In linear environments, hierarchical scoping might require extra discipline from users (like me) that appreciate the cut-and-paste mode of operation.

Personally I like the idea of linear scoping for code files. In (Al)ProTex I addressed this concept with two features: a feature to clear all the existing definitions of code (option `ClearCode'), and a feature for decomposing indexes. Normally I tie the above features with top-level sectioning commands, like \Part or \Chapter, of prose. (Note: indexes that are produced within ProTeX have poor outcome and are big time consumers.)

From: **Stuart Ferguson**
Date: 19 Nov 1993

Chris Nadovich writes: The only unanswered criticism was the issue of scoping. Every time I try to merge a perfectly good piece of code with another---even if they are targeted for separate files---I have to worry about name collisions. The ability to abbreviate scrap names makes this worse.

I tend to agree. One of my first thoughts when finally understanding the design of WEB was, "My God, everything's globally scoped!" How 1970's-ish.

That said, and my language design bias out in the open, the real issue for literate programming is making the scoping decisions understandable to the reader. Where the programmer is comfortable with micro scoping, I think any normal reader will be find a more large scale scoping easier to cope with. I would suggest scoping by section or chapter only, for example, and not in smaller units. I might also insist that the introduction to any scoped section list the public tags and certainly highlight them in a more bold font in the text. Limiting scoping blocks to sections seems like a reasonable compromise between providing the programmer with a powerful tool and keeping the document readable. There are plenty of ways to lay out the index and cross reference to make clear the locality of

reference.

I like all my C files to have the same basic format. Why do I have to rename all the sections in the top-level template when I apply it to a new module. This is unneeded work and it's error prone.

This ties into one of the ideas I've had which I posted here recently, about boilerplate and literate macros. If your web tool had text substitution macros which could be instantiated forward and backwards, you would only need to define your template once and could envoke it with a different module name for each instance. This would obviate the need for scoping in this case.

From: **Michael Koopman**
Date: 19 Nov 1993

Eitan Gurari wrote: Hierarchical scoping of code fragments seems to me to be the way to follow in two-dimensional (e.g., hypertext-oriented) environments that exhibit these relationships in a natural manner. In linear environments, hierarchical scoping might require extra discipline from users (like me) that appreciate the cut-and-paste mode of operation.

Is the underlying issue reuse of literate code (here, within the same web)? Does it seem likely that collections of chunks of a web that describe and implement a reuse object could be written such that the natural language descriptions, method names, attributes, etc., could be defined to allow reuse/renaming? That is, does it seem reasonable to pursue reuse of literate code above the source code cut and paste level; i.e., classes of fully literate code? How close can ADA be considered to approach such capacity without literate support tool?

Has anyone written a literate code segment such that the description and code fragments incorporate named macros that can be renamed to implement an equivalent class in a different domain without needing to overhaul the code or descriptive text? Seems like a library of these things could be powerful medicine. A web tool that supports "common" weave structures could be used to place the various chunks into the desired locations in the web for which they have been decomposed. Has anyone achieved enough familiarity and experience with literate techniques to support or deny the potential for such speculative methods? At a more practical level, inclusion of macro text in the chunk name may avoid some name collisions.

Stuart Ferguson wrote: Limiting scoping blocks to sections seems like a reasonable compromise between providing the programmer with a powerful tool and keeping the document readable.

This gets my vote, although, especially when maintaining older ("golden") code - global scoping might be helpful?

There are plenty of ways to lay out the index and cross reference to make clear the locality of reference.

Adding a chapter mark and id seems a straightforward approach.

I like all my C files to have the same basic format. Why do I have to rename all the sections in the top-level template when I apply it to a new module. This is unneeded work and it's error prone.

This ties into one of the ideas I've had which I posted here recently, >about boilerplate and literate macros. If your web tool had text >substitution macros which could be instantiated forward and backwards, >you would only need to define your template once and could envoke it >with a different module name for each instance. This would obviate >the need for scoping in this case.

I would add support for module name tagging of macros and other web variable entries to (fully?) obviate scoping.

From: **Christian Lynbech**
Date: 19 Nov 1993

On the issue on mixed file/global scoping. I believe that noweb does a good enough job here. As I see it, you cannot mix the two types of scope, without having some (textual) way to mark something as local and something as global. And having to do this anyway, why not either prefix all local scrap names with the file name, thus making them unique to that file, or if that is too much typing (for instance you need to decide on the file name, rather than having a fixed and short marker), pipe it through a small sed/perl/awk script that takes all this-is-a-local-scrap-marker and substitute the file name. The above scheme can be varied endlessly, but it could be easily implemented in a makefile.

From: **Michael Koopman**
Date: 19 Nov 1993

Yes, it is a non-problem.

From: **P Jensen**
Date: 19 Nov 1993

Stuart Ferguson writes: Any good language would allow use before definition, but since C does not, I always have to jump through

hoops getting valid concepts to run through the compiler.

Hear, hear! This comment was apropos of the need to order type definitions properly, but another instance is that of prototypes. I am currently using CWEB, and am sorely tempted to hack on it so that ctangle notices function definitions and constructs matching prototypes, which would then get emitted at a designated point in the output. I see this enhancement as analogous to the ways in which Knuth's original made up for certain deficiencies of standard Pascal.

From: **Stuart Ferguson**
Date: 20 Nov 1993

Stuart Ferguson wrote [Ordering Dependencies]: One problem that I am noticing is that of final ordering of elements in the output program text. Like most one pass compilers, C requires that objects be declared before they are used. This lends a decidedly "bottom-up" quality to raw C code. I am finding that my literate programs tend to be more "top-down," with declarations of higher level objects before the lower level components which they require. The problem is that if I use some generic refinement like <<Public Data Types>> to collect up my typedef's, for example, then the objects are declared in the wrong order for the compiler.

Tommy McGuirewrites: Pardon me for being terribly dense today, but is this a non-problem? If type Foo depends on the definition of type Bar, shouldn't there be a scrap

```
<<Public Data Types>>=
  <Define Type Bar>
  <Define Type Foo>
```

There seems to be a lot of confusion about what I was trying to say, so I suppose I should clarify. Yes, of course you can explicitly put the declarations in the right order and it will work. So there is no "problem" in the sense that work has stopped because something is not possible. However, when I do this I get a twinge from my finely tuned sense of "rightness." It seems wrong that I have to encode the dependency order in this way.

Of course, I get this same feeling every time I program in C. Any good language would allow use before definition, but since C does not, I always have to jump through hoops getting valid concepts to run through the compiler. I've long since learned to ignore the mental warnings I get when programming in C, and it may be that I'm just seeing them anew when attempting literate C programming. But I think I can make a case for the problem being in the literate programming domain and not just the target language.

Most webs allow code chunks with the same identifier, and the code from all the chunks is collected together to form a single refinement of the same name. The code is ordered in the refinement in the same order that it occurs in the document. It seems to me that sometimes the order is conceptually important and sometimes it is not.

If the refinement represents some linear process, like a series of steps, then the order is important. In this case I would argue that the chunks with the same name will for very natural reasons be together in the prose and will be in the right order.

```
"First we do step 1."
<<Some process>>=
  Step #1

"Now we do the next step, which is 2."
<<Some process>>=
  Step #2

"Finally we do the third step."
<<Some process>>=
  Step #3
```

It would never make sense to cut out step two and paste it into another section since that would take it out of context. So we don't generally have to worry about the steps getting out of order as result of editing the prose.

On the other hand, some refinements which have the same name are just used to collect things of similar type and are conceptually order independent. Code chunks for <<Public Functions>>, <<Local Typedefs>> and <<Private Include Files>> will be strewn throughout the web source willy nilly, used whenever a new thing of that type is to be inserted into the output code. When the reader encounters a chunk like

```
<<Include Files>>=
  #include
```

he is not concerned with what comes before or after this chunk in the refinement. The chunk is stand-alone in that it declares only what type of code the chunk represents, not its order in any sequence. If the web is complex, there may be many sections each with many such chunks, and each section is semi-independent in that it could very easily be moved to make the prose read better.

The problem is that even though this type of chunk is conceptually order independent, it is actually order dependent, since the code will be generated in the order it's found in the web. So, if A depends on B, and B comes first in the document, the generated code will compile. If, due to a conceptual restructuring of the web, A is moved to before B, then the generated code will not compile. Now the person who was just making the web read better has to write "glue" code to skick B back in front of A in the code sequence. And where does this tacky glue stuff go anyway? Is it part of the section on A, or the section on B? What if a third thing depends on A or B -- now where does the glue belong?

I would find it conceptually cleaner to allow code chunks to declare their dependencies and be truely independent of the order they occur in the web. We could have:

```
<<Typedefs>>=
  typedef {... B C } A;
  <: defines A and depends on B and C :>

<<Typedefs>>=
  typedef {... C } B;
  <: defines B and depends on C :>

<<Typedefs>>=
  typedef {...} C;
  <: defines C :>
```

These three chunks could be anywhere in the document in any order and if the <<Typedefs>> refinement had the right behavior associated with it, then the chunks which define it would be shuffled into C B A order, which will compile. Even better, if a new type was added it would be shuffled into the sequence correctly and none of the other parts of the web would be affected.

I hope this is more clear now. I think this is really a literate programming problem since it is about making the description of the code clear and elegant and not just about hacking something together that works.

From: **Jeffrey McArthur**
Date: 21 Nov 1993

Stuart Ferguson writes: I would find it conceptually cleaner to allow code chunks to declare their dependencies and be truely independent of the order they occur in the web. We could have:

```
<<Typedefs>>=
  typedef {... B C } A;
  <: defines A and depends on B and C :>
```

The original Pascal WEB has this feature. It understands Pascal and orders the declarations in such a way that simple dependancies are resolved by tangle. If the dependancy is circular, it needs help. All in all, the original Pascal WEB is not that bad of a language to use. Far from perfect. Many of its flaws are do to the nature of Pascal.

From: **Andrew Mauer**
Date: 22 Nov 1993

I like all my C files to have the same basic format. Why do I have to rename all the sections in the top-level template when I apply it to a new module. This is unneeded work and it's error prone.

This ties into one of the ideas I've had which I posted here recently, about boilerplate and literate macros. If your web tool had text substitution macros which could be instantiated forward and backwards, you would only need to define your template once and could envoke it with a different module name for each instance. This would obviate the need for scoping in this case.

Michael Koopman said: I would add support for module name tagging of macros and other web variable entries to (fully?) obviate scoping.

IMHO, this is an excellent idea. I'm interested in hearing the advantages/disadvantages of allowing a user to specify a the module by doing:

```
  @module << C++Class >>
or maybe
  @scope << C++Class >>
```

It seems to me that in the spirit of WEB delimiting of text and program scraps, one would not need to support more than one level of scoping, and a scope should just go on until the next scope is encountered.

What is the real-life situation here? Does anybody have a code that they think needs nested scoping? I agree that it would be more elegant, but for c/c++ (my area), the program structure does not nest enough for that to be likely. I gather that in scheme or other languages which support prodecures-within-procedures this could be important, but is that good style?!?

Reuse of literate programs

From: **Mark Probert**
Date: 18 Nov 1993

I got to thinking about using literate programming tools with the more modern languages, such as C++ and Modula-2, and wondered, "Does anyone have any experience with reuse of literate programming code?" When I code in "straight" C++, for instance, the class definition is in the header and the implementation is in .cc file. But when I use an literate programming tool, both of these are in the same file, as I believe it should be. Now, at reuse time, I have either to use the generated header, which may be "unreadable" (CWEB), or look at the web, which has implementation details that I shouldn't see. The problem appears to be a contradiction between information hiding and correctly documenting code. Can anyone shed any light, or have any opinions :-), on this issue?

From: **Zdenek Wagner**
Date: 18 Nov 1993

I got to thinking about using literate programming tools with the more modern languages, such as C++ and Modula-2, and wondered, "Does anyone have any experience with reuse of literate programming code?"

As I wrote as a response to another's posting, if I want to have some code reusable, I design my private library (functions, class implementation). Then I need the .h files and .lib file. Somewhere I have the printed vowen file so that I can refer to it.

From: **Stuart Ferguson**
Date: 18 Nov 1993

When I code in "straight" C++, for instance, the class definition is in the header and the implementation is in .cc file. But when I use an literate programming tool, both of these are in the same file, as I believe it should be. Now, at reuse time, I have either to use the generated header, which may be "unreadable" (CWEB), or look at the web, which has implementation details that I shouldn't see.

I've been dealing with this two different ways. The simplest is just to organize the document for reuse with a chapter on the public interfaces and how to use them and other chapters on the implementation. The idea is to make the document readable by a client of the package without letting the details get in the way. In many OO systems it is considered normal for clients to look at the implementation too, so having it there may be good.

Another approach would be to generate two different forms of your web -- one for the client and one for the maintainer. You might use "\if" or some other mechanism to generate a version of the document with only the public interfaces and no implementation. This is like the Eiffel short form, and has the advantage that the long form may contain public interfaces anywhere in the document which will be collected together for the client to see.

From: **Lee Wittenberg**
Date: 18 Nov 1993

Mark Probert writes: I got to thinking about using literate programming tools with the more modern languages, such as C++ and Modula-2, and wondered, "Does anyone have any experience with reuse of literate programming code?" When I code in "straight" C++, for instance, the class definition is in the header and the implementation is in .cc file. But when I use an literate programming tool, both of these are in the same file, as I believe it should be. Now, at reuse time, I have either to use the generated header, which may be "unreadable" (CWEB), or look at the web, which has implementation details that I shouldn't see. The problem appears to be a contradiction between information hiding and correctly documenting code. Can anyone shed any light, or have any opinions :-), on this issue?

What I do (when I'm diligent) is to generate manual pages for the class within the web (usually in "limbo," before the first section). That way I can maintain the manpages along with the code, but still xerox them separately for users. In noweb, I usually have a \part{Interface}, which contains the manual pages, followed by a \part{Implementation}, which contains the web proper. I haven't yet done any C++ programming in noweb, so I'm not sure where I would put the class declaration, but I suspect I would keep it in the Implementation part, saving Interface for purely human issues. Yet another technique stolen from Steve Avery.

CWEB thoughts

From: **Phil Jensen**
Date: 03 Dec 1993

I just got _The Stanford GraphBase_, and I'm terrifically impressed, as with everything else DEK has written. Some thoughts on CWEB, and one or two questions for anyone who has looked at the sources (I don't have the time right now). I have considerable distaste for C. Among the things I wish had been done differently are: the meaning of = and ==; declaration and typename syntax; open-ended control constructs (vs. Modula's if...end, while...end etc.); use of int for Boolean; line-oriented preprocessor; absence of |var| parameters; treatment of arrays as pointers...

But as I read The SGB, I'm thinking - this is okay, I can live happily with this. Just as I believe that literate programming is far more significant than OO, I also believe it is more important than details of language syntax. (Semantics are different - I still want RAISE, TRY..EXCEPT, TRY..FINALLY, and LOCK..DO.) I'm also using CWEB, and am evolving some guidelines:

- Use a semicolon after every fragment reference occurring in a statement-like context;
- Enclose the body of every statement-like fragment in {...} except when it looks silly, as for things like "case N: stmts";
- Always use {...} after an |if| which is followed by an |else| (and never put an |if| and a matching |else| in different fragments!);
- Otherwise, don't use {...} with control constructs which govern a single statement or single fragment reference, EXCEPT
- Do use {...} if the enclosed fragment has multiple defining occurrences (usually things like @<Initialize...@> and @<Handle cases...@>.)
- [Disclaimer - all this, like indentation, has elements of religious preference - I just post these guidelines for general interest.]

A couple questions, if anyone knows, vis-a-vis CTANGLE: Knuth gets |struct foobar| with both words in boldface, but I get the tag in italic---why? Knuth seems to get no line break between { and a declaration, but I do---why? Thanks and happy LitProgramming.

From: **Lee Wittenberg**
Date: 05 Dec 1993

Phil Jensen writes: I just got _The Stanford GraphBase_, and I'm terrifically impressed, as with everything else DEK has written. Some thoughts on CWEB, and one or two questions for anyone who has looked at the sources (I don't have the time right now). [text omitted] I'm also using CWEB, and am evolving some guidelines:

- *Use a semicolon after every fragment reference occurring in a statement-like context;*
- *Enclose the body of every statement-like fragment in {...} except when it looks silly, as for things like "case N: stmts";*
- **** Always use {...} after an |if| which is followed by an |else| (and never put an |if| and a matching |else| in different fragments!);*
- *Otherwise, don't use {...} with control constructs which govern a single statement or single fragment reference, EXCEPT*
- *Do use {...} if the enclosed fragment has multiple defining occurrences (usually things like @<Initialize...@> and @<Handle cases...@>.)*

I pretty much agree with all of these, and follow them in my own programs. I would go further with the one I've marked ***, though. I always make sure that each control construct is in a single fragment (with possible sub-fragments, of course, for nested statements). The only exception is in case-like statements: the "case N:" labels and their corresponding statements are almost always in the sub-fragments.

A couple questions, if anyone knows, vis-a-vis CTANGLE: Knuth gets |struct foobar| with both words in boldface, but I get the tag in italic---why?

Either the definition of `struct foobar' is in a C chunk that occurs earlier in the web than the text containing `|struct foobar|, or Knuth has snuck an `@s foobar int' into the web somewhere (`@s' is a silent `@f').

Knuth seems to get no line break between { and a declaration, but I do---why?

He probably puts an `@+' (or a bizarre @t sequence) into the web immediately after the {.

From: **Norman Ramsey**
Date: 05 Dec 1993

A couple questions, if anyone knows, vis-a-vis CTANGLE: Knuth gets |struct foobar| with both words in boldface, but I get the tag in italic---why? Either the definition of `struct foobar' is in a C chunk that occurs earlier in the web than the text containing `|struct foobar|,

or Knuth has snuck an `@s foobar int' into the web somewhere (`@s' is a silent `@f').

Knuth seems to get no line break between { and a declaration, but I do---why? He probably puts an `@+' (or a bizarre @t sequence) into the web immediately after the {.

And people wonder why I dislike prettyprinting :-) :-) :-) :-)

From: **Bart Childs**
Date: 06 Dec 1993

Phil Jensen posed some CWEB questions after reading some of Knuth's Stanford GraphBase. The answer to one question is that Knuth inserted `@+' in many places to supress a line break. The first question is probably that Knuth inserted a format statement like @f foobar struct which says use the same typesetting formatting for foobar that is used for struct.

Phil's guidelines were quite good and remind me of the Strunk and White type rules. Some of the same kinds are imbedded in the FWEB documentation from John Krommes. The sources of GraphBase are available for anonymous ftp from labrea.stanford.edu directory /pub/sgb

Krommes FWEB 1.40 has been interesting to test. It has a LaTeX style and a verbatim language. The latter gives formatting similar to NoWEB and NuWEB in that the HLL code is not messed up. However, I don't like it. The section numbers disappear and show up as part of the section names <Some section name 4.5>=. I am stating my strong preference for the section numbers appearing at the beginning of the section. If a variable is referenced in the documentation part, you have to find a code part to be able to find the bounds of the section. This is particularly troublesome if you have sections that are documentation only. Still, I am grateful for the contributions of John, Norman Ramsey, Preston Briggs, ...

From: **Zdenek Wagner**
Date: 06 Dec 1993

Norman Ramsey said: And people wonder why I dislike prettyprinting :-) :-) :-) :-)

I still insist on prettyprinting. If I look at my program a year later, I know that the boldface objects are either C++ keywords or my classes/structures. Since |foo| is not a C++ keyword, it defined somewhere as class/structure. Thus I can see more things at the first glance :-)

From: **John Krommes**
Date: 08 Dec 1993

Bart Childs writes with regard to FWEB 1.40: Krommes' FWEB 1.40 has been interesting to test. It has a LaTeX style and a verbatim language. The latter gives formatting similar to NoWEB and NuWEB in that the HLL code is not messed up.

However, I don't like it. The section numbers disappear and show up as part of the section names <Some section name 4.5>=. I am stating my strong preference for the section numbers appearing at the beginning of the section. If a variable is referenced in the documentation part, you have to find a code part to be able to find the bounds of the section. This is particularly troublesome if you have sections that are documentation only.

That section numbers do not appear at the beginning of the section in FWEB 1.40 is only a default. It is very simple to make them reappear: just put \numberTeXtrue in the limbo section. For most programming purposes, I agree with Bart; I like the section numbering to be explicit, for precisely the reason he mentions about sections that are documentation only. However, perusing my request queue, for every user who agrees with Bart, there's one who passionately wants the documentation to look more "book-like." FWEB 1.40 (which is still highly experimental) attempts to let the user choose.

FWEB 1.40 was announced to the FWEB mailing list but not here because I don't want people, and especially large programming projects, dependent on a stable FWEB to convert just yet. However, I do need patient volunteers to experiment. It's available via anonymous ftp from ftp.pppl.gov:/pub/fweb

---

C++ API Extractor

From: **Paul Bandler**
Date: 21 Dec 1993

I am new to reading this group so appologies for what is I'm sure a very basic question. We're at the begining for a C++ development and establishing coding / documentation practices for the project. We would like to be able to build a high quality (i.e. to be distributed

as part of a software product) programmer's reference manual derived from the C++ source files. In investigating this I have been lead to this news group and understand that there is a program called CWEB which may be of assistance to us.

Could someone advise:

1. Whether CWEB is a tool suitable to my purpose?

2. Is CWEB available in the public domain at an archive site? If so, where?

3. Would someone be willing to send me an example CWEB input and output so that I can get an idea of what is involved?

4. Would it be possible to integrate CWEB with Framemaker?

5. There is some resistance here to the effect of needing to instrument ones source code extensively in order to be able to create product quality end-user documentation from it. Can anyone comment as the feasibility/desirability of this approach or indicate whether this has been achieved successfully in practice somewhere?

From: **Lee Wittenberg**
Date: 23 Dec 1993

Paul Bandler writes: I am new to reading this group so appologies for what is I'm sure a very basic question.

No apologies necessary. That's what the group is for (IMHO).

We're at the begining for a C++ development and establishing coding / documentation practises for the project. We would like to be able to build a high quality (i.e. to be distributed as part of a software product) programmer's reference manual derived from the C++ source files. In investigating this I have been lead to this news group and understand that there is a program called CWEB which may be of assistance to us. Could someone advise: 1. Whether CWEB is a tool suitable to my purpose?

Yes. CWEB 3.0 (and thus the current version 3.1, as well) is designed to work with C++. I have used it for some C++ work myself.

2. Is CWEB available in the public domain at an archive site? If so, where?

The official site is labrea.stanford.edu. I believe it is in the ~/pub/cweb directory.

3. Would someone be willing to send me an example CWEB input and output so that I can get an idea of what is involved?

CWEB itself is just such an example, but if you'd like something simpler, I have a few sample webs available in the ~/pub/leew/samples.LP directory of bart.kean.edu.

4. Would it be possible to integrate CWEB with Framemaker?

It ~may~ be possible, if you're willing to do the rewriting, but CWEB is set up (designed?) to work with TeX.

5. There is some resistance here to the effect of needing to instrument ones source code extensively in order to be able to create product quality end-user documentation from it. Can anyone comment as the feasibility/desirability of this approach or indicate whether this has been achieved successfully in practice somewhere?

Literate programming doesn't really work well for end-user documentation (IMHO). What it ~does~ do is to turn the program itself into a high-quality programmer's reference (which is what you asked about in the first place).

Importance of indexing?

From: **Mark Carroll**
Date: 18 Jan 1994

I'm currently working on an HTML based literate programming tool. (For those who don't know, HTML is an SGML based markup language for hypertext documents. HTML can be read by the Mosaic tool from NCSA, or with a variety of other tools including an

emacs mode.) I've got the tool working, but I don't do any indexing of variable definitions. I haven't implemented that, because I *never* look at the things when I read a literate program. (They wouldn't be too hard to add, but the addition of the index would involve some rather un-SGML like contortions, which I've been trying to avoid.)

So, at last, my question is: how important do other, more experienced literate programmers find indentifier indices? (For interested persons: Yes, I do plan to release this tool eventually. However, it's currently rather ugly, and needs serious work before anyone else sees it. I've currently got a version that's just capable of bootstrapping itself. No one but me gets to see it until the code is considerably cleaned up.)

From: **John Lacey**
Date: 18 Jan 1994

*Mark Carroll writes: I've got the tool working, but I don't do any indexing of variable definitions. I haven't implemented that, because I *never* look at the things when I read a literate program. [...]. So, at last, my question is: how important do other, more experienced literate programmers find indentifier indices?*

I don't find the index useful at all, *UNLESS* it is on each page, as in the published TeX and METAFONT programs. In that case, I find it indispensable. (I have a love/hate relationship with the programs themselves, but I will not deny either their readability or the skills of the author.)

IMHO, WEB and it's ilk are useless without indexing. If you remove the features that simply cater to a deficient language, those which can be handled with sed or awk one-liners (OK, maybe several-liners), and those that vgrind/tgrind handle, you are left with just indexing (and, admittedly, the ability to use and typeset really long procedure names in the local vernacular).

From: **Andrew Mauer**
Date: 18 Jan 1994

Mark Carroll said: So, at last, my question is: how important do other, more experienced literate programmers find indentifier indices?

I certainly DO NOT QUALIFY ... but I have "tangled" with noweb (as my rather questionable posts to this group have attested), and I have found identifier indices *very nice*. If one is trying to change just a part of a program, easy access to the "information flow" allows you to tweak it without understanding the whole structure. This greatly increases the ease of hacking (and presumably maintenance). :-)

From: **Lee Wittenberg**
Date: 18 Jan 1994

Mark Craig Carroll asks: So, at last, my question is: how important do other, more experienced literate programmers find indentifier indices?

I find them invaluable. Not for my own code -- I usually know where things are, but when I'm reading someone else's web, I find myself constantly referring to the indentifier index: ``How and where is that d---ed variable (or function) declared?''. "The Stanford GraphBase" would be an order of magnitude harder to get through if the identifiers weren't indexed, and I don't think I ever would have gotten CWEB 2.4 working on my PC if not for the identifier index.

(For interested persons: Yes, I do plan to release this tool eventually. However, it's currently rather ugly, and needs serious work before anyone else sees it. I've currently got a version that's just capable of bootstrapping itself. No one but me gets to see it until the code is considerably cleaned up.)

I know the feeling.

From: **Stuart Ferguson**
Date: 19 Jan 1994

*Mark Carroll) writes: I'm currently working on an HTML based literate programming tool. [...] I've got the tool working, but I don't do any indexing of variable definitions. I haven't implemented that, because I *never* look at the things when I read a literate program. (They wouldn't be too hard to add, but the addition of the index would involve some rather un-SGML like contortions, which I've been trying to avoid.)*

Just a thought -- but doesn't HTML support a form of hypertext? If so, why not have the symbol references be linked back to their definition site?

From: **Preston Briggs**
Date: 19 Jan 1994

*Mark Carroll writes: I've got the tool working, but I don't do any indexing of variable definitions. I haven't implemented that, because I *never* look at the things when I read a literate program. So at last, my question is: how important do other, more experienced literate programmers find indentifier indices?*

I don't use indices so much for my own code (though I'm finding them more useful as my code gets older); however, we (meaning my group) use them quite a lot in reviewing each others code. In olden times, we wanted a terminal handy when we did walkthroughs, so we could find places where a variables was declared or initialized. It was such a typical walkthrough question to suddenly ask: "Hey, is this thing initialized to NULL (or zero or "blocks" or "blocks - 1" or initialized at all)?"

From: **Trevor Jenkins**
Date: 19 Jan 1994

So, at last, my question is: how important do other, more experienced literate programmers find indentifier indices?

I rarely use the full indices for programs that I've written. However, when reading DEK's source of TeX and metafont I find the mini-indices invaluable.

From: **Bart Childs**
Date: 20 Jan 1994

It is easy to lose sight of the usefullness of indexes if all you do is write new code. They are invaluable. I ported TeX and MetaFont to several machines before volume B of Computers & Typesetting was available where the mini-indices were on the pages you look at. Still, they were invaluable although the mini-indices were there on the last port I did. That too is a big help.

Have you ever taken a course from a textbook being developed? I have not but have taught one several semesters in that situation (no I am not the author either) and it is pure torture for the student.

We often hear that 60 to 80% of the cost of software in its lifetime is maintenance. If that is true, then every help we can give the maintenance programmer should lead to economy, profit, or whatever drives your system. I think the aids of keywords, programmer supplied names, ... in the formatting of the language specific WEBs and their indices (or indexes) are two great helps for the future maintainer of my software. The structure and integration of documentation are also valuable.....

From: **Lee Wittenberg**
Date: 20 Jan 1994

Bart Childs writes: Have you ever taken a course from a textbook being developed? I have not but have taught one several semesters in that situation (no I am not the author either) and it is pure torture for the student.

I don't remember it being so bad. On the other hand, it was in a small seminar rather than a large lecture. That may have made the difference.

We often hear that 60 to 80% of the cost of software in its lifetime is maintenance. If that is true, then every help we can give the maintenance programmer should lead to economy, profit, or whatever drives your system. I think the aids of keywords, programmer supplied names, ... in the formatting of the language specific WEBs and their indices (or indexes) are two great helps for the future maintainer of my software.

Hear, hear!

The structure and integration of documentation are also valuable.....

Ditto, ditto!

From: **Mark Carroll**
Date: 20 Jan 1994

Stuart Ferguson writes: Just a thought -- but doesn't HTML support a form of hypertext? If so, why not have the symbol references be linked back to their definition site?

Well, strictly speaking, HTML doesn't *support* a form a hypertext; it *is* a form of hypertext. The reason that I'm avoiding the idea of linking symbol references back to their definition sites is because for the applications I'll be using this for, it doesn't make any sense. Linking symbols to definitions only works if there is exactly one definition for each symbol - and that's not the case in programs that I write. I do most of my work in object-oriented languages, where a given symbol will have a list of definition sites.

My reason for being biased against use indexing is related to this. If each symbol has multiple definitions, then a naively generated use index is going to lump together all uses of a given name together, even though they are references to different definitions. And there's no good way to automatically determine which definition is being referenced by a given use, without either giving the weaver knowledge about the language, or requiring the programmer to provide the information. Since I don't want a language specific system, that leaves me with forcing the user to annotate uses.

A little bit of toying around leads me to believe that annotating uses is cumbersome. But I'll probably give it a try, since so many people seem to think it's very important. How's this: by default, I assume that any uses of an identifier defined within the default index of a chunk will be references to the definition in that index. If the identifier is defined once within the index, then it can even insert a hyperlink to the definition. If it's outside of the current index, then we need some way to identify where we expect uses to come from, so the identifier gets marked in a u (for use) tag, which has an index field, as in the following:

```
<chunk name="blah" index="blech">
   for (<u index="foo">loopindex</u> =0; loopindex<100; loopindex++)
      <cr cref="bar">
</chunk>
```

A question that naturally comes up here is: should all uses of an identifier be assumed to be interesting, or should the programmer mark interesting uses? That should take care of most things. Thinks with multiple definitions in an index can't get an automatically generated hyperlink unless I add some way of identifying the correct one. Frankly, I think that that's just overkill and not worth the effort. I'll always insert a link to the currently defined index at the end of the chunk, which you can follow, and check out the multiple links that occur in the index.

From: **Reino de Boer**
Date: 21 Jan 1994

First of all, let me say that I cheer the use of HTML for literate programming. And, I like the idea of having more than one index. In nuweb I use nuweb's indexing for user-identifiers, and LaTeX's indexing for standard identifiers. For statistics: I'm using nuweb (and some cweb).

Mark Carroll writes: Linking symbols to definitions only works if there is exactly one definition for each symbol - and that's not the case in programs that I write. I do most of my work in object-oriented languages, where a given symbol will have a list of definition sites.

Maybe the Next/Previous tags, that are being suggested for HTML+, can be used to create a (virtual) list of definitions to browse.

Format of literate programming bibliography

From: **Pat Pinchera**
Date: 18 Jan 1994

I recently obtained the Literate Programming Bibliography via anonymous ftp. I just have one question: what format is this file in, and where can I get the necessary s/w to read or translate this file? If you know of a FAQ or ftp site to get the files, (I assume Tex formatter or something), just tell me that. I am very interested in using literate programming to document my C++ class libraries. Thanks.

From: **Lee Wittenberg**
Date: 18 Jan 1994

Pat Pinchera writes: I recently obtained the Literate Programming Bibliography via anonymous ftp. I just have one question: what format is this file in, and where can I get the necessary s/w to read or translate this file? If you know of a FAQ or ftp site to get the files, (I assume TeX formatter or something), just tell me that. I am very interested in using literate programming to document my C++ class libraries. Thanks.

It's in BibTeX format. BibTeX is usually distributed as part of the LaTeX package. If you already have access to TeX, you probably already have access to BibTeX. If not, you will probably want to get a copy of TeX from the CTAN archives (probably the same place you got the bibliography). Which TeX implementation you choose will depend on the operating system you're using. If it's DOS or OS/2, I can highly recommend emTeX, which definitely comes with BibTeX.

Literate hypertext indices

From: **Mark Carroll**
Date: 19 Jan 1994

Greetings! The response to my earlier question about the importance of indexing has been overwhelmingly in favor of including indices. A large number of those have requested multiple indices, rather than a single indentifier index at the end of the document. In response to this, I've come up with a tentative idea for how to handle indices, which I'd like to open up for comments. First, let me introduce the syntax of my system, which should allow me to make a little more sense when I explain the indexing idea. Chunks in my system are described using an SGML style begin/end tag. The begin tag takes two fields: a mandatory name field, and an optional file field. If the file field is present, the chunk is a root chunk, and will be tangled to the file specified in the field argument. Chunk references are simple tags with a reference field, naming the chunk. Eventually, this reference will be allowed to link to another file, but my tangler doesn't handle that yet. An example chunk with references:

```
<chunk name="a sample chunk" file="sample.c">
#include <stdio.h>

main() {
   for (i=0;i<10;i++)
      <cr ref="another chunk">
}
</chunk>
```

Now, the indexing proposal that I've come up with fits in with this syntax. The chunk opener will take an extra field, specifying the index into which definitions from this chunk will be inserted. Then, definitions are declared using a definition tag around the identifiers to be indexed. So:

```
<chunk name="a sample chunk" file="sample.c" index="mainindex">
#include <stdio.h>

<d>main</d>() {
   int <d>i</d>;
   for (i=0;i<10;i++)
      <cr ref="another chunk">
}
</chunk>
```

The definition tag will have an optional field naming an index to insert the identifier into if it is not the default index for the chunk. The weaver will generate an index file for each specified index. The file author can provide links into this file from wherever it is appropriate to reference an index. I'm inclined towards only indexing the definitions of an identifier, and not uses. But I could probably be convinced to add use indexing as an option, if enough people prod me. Does this seem like a reasonable indexing method for a hypertext web?

From: **Mark Probert**
Date: 20 Jan 1994

Mark Carroll writes: First, let me introduce the syntax of my system, which should allow me to make a little more sense when I explain the indexing idea.

I may have this wrong, but are you developing a WEB system that uses a modified SMGL/HTML as its language? As a personal observation, I find the syntax you are preposing a little cumbersome compared to nu/noweb. Would it not be better to use one of these

"standard" tools and re-target the output to HTML? If you made this a switch, then you would have the benefit of being able to produce a hardcopy in LaTeX and have a hypertext version in HTML. I had a quick look at doing this in nuweb, but Preston designed the output sections with LaTeX in mind. Maybe noweb would be an "easier" target.

From: **Jacob Nielsen**
Date: 20 Jan 1994

Mark Carroll writes: Greetings! The response to my earlier question about the importance of indexing has been overwhelmingly in favor of including indices. A large number of those have requested multiple indices, rather than a single indentifier index at the end of the document.

I second the use of multiple indices --- in principle. The problem with multiple indices is that they are so hard to get right the first time (just like ordinary indices). All to often you end up with a lot of unnessesary identifiers; I speak from experience and that was with just one index :-(

In response to this, I've come up with a tentative idea for how to handle indices, which I'd like to open up for comments.

I like the idea (if only I could control myself with such a tool :-)

I'm inclined towards only indexing the definitions of an identifier, and not uses. But I could probably be convinced to add use indexing as an option, if enough people prod me.

Unless you're doing object-orientated programming, I'd say you have to provide some indexing of identifiers in interesting places. BTW: OOP adds to your basic indexing of definitions and I guess it breaks your hypertext system as it is (I hope not). If we have:

```
for(i=0;i<list.lastIndex;i++)
  (Frame *)( list.get(i) )->draw();
```

the Frame class doesn't define the draw method, but it's inherited from the GraphicBase class. I would like to tell me reader that Frame::draw is intended to be called (conceptually good idea) but due to subclassing the draw method in GraphicsBase is called. Opinions, anyone??

Does this seem like a reasonable indexing method for a hypertext web?

Yes, if the programmer doesn't get carried away, thus making the resulting web more or less unreadeble. If I was using (i.e. reading) a hypertext web, I would expect to be able to pick on one identifier and from there be able to see: 1. Where it's defined. 2. Interesting places where it's used. --- this is where the fun part of indexing starts :-) 3. All places where it's used.

From: **Andrew Dobrowolski**
Date: 20 Jan 1994

Mark Probert writes: I may have this wrong, but are you developing a WEB system that uses a modified SMGL/HTML as its language? As a personal observation, I find the syntax you are preposing a little cumbersome compared to nu/noweb. Would it not be better to use one of these "standard" tools and re-target the output to HTML? If you made this a switch, then you would have the benefit of being able to produce a hardcopy in LaTeX and have a hypertext version in HTML....

Agreed, however, the syntax proposed by Mark Carroll would not be awkward in a native SGML editor where markup does not appear as a sequence of ascii characters. In such an editor Mark's system would be an very good way to write literate code. Using noweb or nuweb would be awkward in comparison because the advantages of SGML, for example the separation of content (the code) from structure (the noweb or nuweb directives), would not be applied.

As for the need for an index, it may not be necessary for online use with an appropriate editor, but in my limited experience with literate programming I found it useful for reading printed code.

From: **Lee Wittenberg**
Date: 20 Jan 1994

Mark Carrol has presented a tentative syntax for his SGML-based literate programming system, requesting comments on the syntax. I don't know SGML, but the proposed syntax seems like a fairly straightforward extension of what he's already got, which (IMHO) is all

to the good. The simpler the syntax the better (that's why I like noweb).

I'm inclined towards only indexing the definitions of an identifier, and not uses. But I could probably be convinced to add use indexing as an option, if enough people prod me.

Indexing uses is vital if you're maintaining a program you didn't write! We can't count on the people who come before us to use what we consider a completely modular design :-(, so there is no guarantee that identifer use is encapulated in any meaningful way (encapsulation itself causes some interesting problems in webs, but that's another story). The best way to find out the consequences of messing with a variable or function is the check in the index to find all the places where the appropriate identifier is used (it helps to find bugs, too). Knuth's mini-indices are nice, but they can't help here. noweb's mini-indices include pointers to where identifiers are used, but I still find myself relying mostly on the main index (of course, this may simply be force of habit).

I should mention that I am one of those people who prefers to spend as much time as possible off of the computer. I therefore prefer to work from a printed, woven listing. This certainly colors my opinions. Indices are not needed at all when one sits at the computer with the source (that's what grep is for).

From: **Peter Johnson**
Date: 20 Jan 1994

Mark Probert writes: As a personal observation, I find the syntax you are preposing a little cumbersome compared to nu/noweb. Would it not be better to use one of these "standard" tools and re-target the output to HTML?

To someone familiar with SGML, this is probably amusing. What we do when we write literate programs is "markup" the text (code). Since SGML is a standard for text markup (ISO8879) I would tend to call nuweb and noweb the nonstandard tools.

If you made this a switch, then you would have the benefit of being able to produce a hardcopy in LaTeX and have a hypertext version in HTML. I had a quick look at doing this in nuweb, but Preston designed the output sections with LaTeX in mind. Maybe noweb would be an "easier" target.

I have been playing with literate programming and generating HTML output. I have looked at both ways, and find using SGML as a starting point to be preferable. One nice thing about using SGML for literate programming is that you can develop some simple tools, and then incorporate literate programming into any type of document that you like. For example, if you wanted HTML output, you could come up with a new document type for HTML which has literate programming extensions. Then with minimal fuss you can convert this into HTML using an existing SGML parsing tool. This is what I have been playing with. On the other hand, if you wanted to output TeX, you could use the same literate programming extensions with sgml documents designed to generate TeX.

As for syntax being complicated. I have to agree, but if you use an SGML parser, you can use a number of SGML markup minimization techniques, and have the parser normalize the output.

```
<chunk name="a sample chunk" file="sample.c" index="mainindex">
#include <stdio.h>

<d>main</d>() {
   int <d>i</d>;
   for (i=0;i<10;i++)
      <cr ref="another chunk">
}
</chunk>
```

becomes

```
<chunk name="a sample chunk" file="sample.c" index="mainindex">
#include <stdio.h>

<d/main/() {
   int <d/i/;
   for (i=0;i<10;i++)
      <cr/another chunk/
}
</chunk>
```

Of course, there are also ways of getting around specifying the file and index attributes of the chunk.

From: **Mark Probert**
Date: 21 Jan 1994

Agreed, however, the syntax proposed by Mark Carroll would not be awkward in a native SGML editor where markup does not appear as a sequence of ascii characters. In such an editor Mark's system would be an very good way to write literate code.

Are native SGML editors freely available? I am using HTML-mode in emacs writing for Mosiac and don't have a full SGML "environment". So, for me, a hypertext document is a ASCII document full of <xx> commands.

From: **Andrew Dobrowolski**
Date: 21 Jan 1994

Mark Probert writes: Are native SGML editors freely available? I am using HTML-mode in emacs writing for Mosiac and don't have a full SGML "environment". So, for me, a hypertext document is a ASCII document full of <xx> commands.

I don't know of any public domain native SGML editors although a number of commercial SGML editors exist on a variety of platforms and most capable of supporting any SGML literate proggraming scheme. For a long time I've been thinking about (sigh... not doing anything about) hooking one of the public domain SGML parsers into Emacs and using read-only zones to represent the <xx> and </xx> tags that mark-up document structures. With a fair deal of custom mode work an Emacs user would have a nice environment for editing SGML without having to learn a whole new editor. Maybe some kind soul has already done this. I just don't know what's out there right now.

From: **Norman Ramsey**
Date: 24 Jan 1994

```
<chunk name="a sample chunk" file="sample.c" index="mainindex">
#include <stdio.h>

<d>main</d>() {
   int <d>i</d>;
   for (i=0;i<10;i++)
      <cr ref="another chunk">
}
</chunk>
```

This kind of markup is useful only if it can be hidden from users by a suitable editor. I applaud any attempt to build such an editor, provided details like this can be hidden from those who don't care to tinker with the implementation. One suggestion: if you can map existing web files (noweb, nuweb, CWEB, fweb, or FunnelWeb) into your notation, people will be more willing to try out your editor. Only noweb is designed to make this exercise easy, but it shouldn't be too bad with nuweb because the feature set there is pretty small. For extra bonus points, make it possible to map in the opposite direction also. It might be worth studying Marcus Brown's work to find out what's already been done in the realm of interactive support for literate programming.

SGML and software development

From: **Steve Heaney**
Date: 20 Jan 1994

An few thoughts that I'd appreciate some input on please. It occurs to me that SGML (the Standard Generalised Markup Language) provides many possibilities for supporting a software development environment. Because SGML provides the means of defining and verifying the content and structure of a "document" it offers, for example, the potential for coupling the documentation produced by software process (requirements, spec, design docs) with program source and beyond to maintenance manuals.

This could take the form of a literate programming tool such as Web which combines source and documentation into one file. Alternatively it could form the basis of a more comprehensive CASE environment. Once you start thinking about it, many opportutities to exploit SGML occur. How about:

- being better able to track requirements through specification, design and implementation.

- supporting hypertext in your "source code".
- generating or validating interfaces directly from your documentation.
- formal validation of a program against assertions in the documentation.
- defining a high level programming language which can be transformed into a number of target languages.

Of course, none of these suggestions is immediately provided as an intrinsic feature of SGML - but SGML has the potential to provide an extensible, vendor neutral, language independent environment for software development with a few carefully thought out DTD's and the use of some of the software available on the net. Maybe someone has tried some of this already. I would be very interested if anyone has DTD's or DTD fragments for program specification or design, especially if these have been coupled with tools for generating or validating the resultant source code.

From: **C. M. Sperberg-McQueen**
Date: 20 Jan 1994

Steve Heaney writes: Maybe someone has tried some of this already. I would be very interested if anyone has DTD's or DTD fragments for program specification or design, especially if these have been coupled with tools for generating or validating the resultant source code.

There is an SGML DTD fragment for the specification language Z, which is part of the Z Interchange Format (contact John.Nicholls@prg.oxford.ac.uk for more information) and there are a few people, myself included, who have thought and sometimes talked about writing a DTD or DTD fragment for use in an SGML-based literate programming system. But I don't know of any finished systems for general programming.

(I do use a finished system, inspired by Web, to document SGML document type declarations. But the next revision will, I think, be much more like nuweb and noweb, and the scrap mechanism will look a lot like what Mark Carroll describes, with the addition of a 'lang' attribute to identify the language of a scrap, so that language-specific routines can reliably be invoked, to handle indexing or pretty-printing or what have you.)

It is not necessarily the case, of course, that the entire evolution of a system, from specification to code, should be documented in a single document. But of course it would be handy to do specifications documents, user manuals, and literate programming using the same document processing tools. Those who use TeX or LaTeX for everything can already do this; those of us who prefer SGML still need the tools. The first cut, of course, can be a simple translation from the SGML tag set into the existing web system of one's choice.

From: **Eliot Kimber**
Date: 21 Jan 1994

Steve Heaney writes: An few thoughts that I'd appreciate some input on please. It occurs to me that SGML (the Standard Generalised Markup Language) provides many possibilities for supporting a software development environment.... Maybe someone has tried some of this already. I would be very interested if anyone has DTD's or DTD fragments for program specification or design, especially if these have been coupled with tools for generating or validating the resultant source code.

There are several products that are at least potentials for this, including Passage Systems' PassagePRO, MicroStar's Computer Aided Document Engineering (CADE) system, InfoDesign's WorkSmart, and Documentum's integration platform.

Certainly the application of web-type approaches to code and documentation management is a natural for SGML. In fact we did this (sort of) to develop the IBMIDDoc DTD by making the source for the DTD primarily the source for the IBMIDDoc Reference and then creating a process that builds the working DTD from the documentation source (unfortunately, we didn't have the necessary SGML tools in place when we started so right now it's a DCF/Script process, not an SGML process).

When I talk about designing SGML applications for technical information, I present the task as essentially one of relating information objects (the SGML elements) to objects in the data model for the system being documented (all real systems have a data models, whether it's been formalized or not, and since technical documentation is always about real systems (possibly systems of concepts), all technical documentation can be thought of as documenting objects in data models). The IBMIDDoc language, for example, uses and supports this approach explicitly.

Once you get this far, that the information is always mapped to a data model, I go one more step and suggest that the information objects are in fact objects in the data model itself. In other words, the information about the system is part of the system itself. This should be obvious in the case of integrated online information like help or tutorials, but I think it works for all information, regardless of media. If you've gotten this far, I think it follows that in a CASE environment, there is no meaningful distinction between code and information and therefore no distinction (at a high level) between the tools used to create and manage code and information. And since SGML is, by its nature, generally the richest information source form (because of its generality), it makes sense that it would be the primary encoding for all data in the system, because everything else can be derived from it.

From: **Jeffrey McArthur**
Date: 21 Jan 1994

The more I deal with SGML the more I hate SGML. I like the concept of SGML. If John Goldfarb had taken just a few classes in language design then SGML would be a wonderful idea. The problem I have is that SGML, in its current form, is such an awful hack. The syntax for dtd's is needlessly complex. SGML has the problem of creeping featurism. Consider the discussions in comp.text.sgml about ambiguous DTD's. Any language that allows you to create a "grammer", a.k.a DTD, that is ambiguous has serious problems.

On thing I really hate about SGML is the way end-of-line is treated. Instead of being "white space" its meaning is context dependant. This means any tool that you use on an SGML file MUST understand the context. Unfortunately, SGML appears to be here to stay. Nothing else seems to match what SGML does. I prefer to work with a small subset of SGML; not the entire hideous beast.

Literate programming is supposed to make programs easier to understand. Can anyone honestly say that the following is easy to understand?

```
: <chunk name="a sample chunk" file="sample.c" index="mainindex">
: #include <stdio.h>
:
: <d>main</d>() {
:    int <d>i</d>;
:    for (i=0;i<10;i++)
:       <cr ref="another chunk">
: }
: </chunk>
```

Besides being ugly it is not even good ANSI C. There is no reason that < and > should be used for tagging. SGML allows you to use other characters. C/C++ programs generally only use the characters in the range [1-127]. This leaves 128 characters in that could be used for tag delimters. Pick any two and use them instead. Personally, I prefer literate programming tools that understand and can pretty print the language.

From: **Robert Lougher**
Date: 21 Jan 1994

There has been quite a few articles posted recently about the use of HTML and SGML in literate programming. It seems as if there is a hell of a lot of interest in using hypertext etc. as a means of augmenting existing literate programming systems. For example, a number of people (sorry I can't remember who) have mentioned using hypertext links to link definitions and usages together. There has also been mentions of the importance of supporting maintenance, as this is the most costly part of the life-cycle. This is where I come in! I've been working on this area towards my PhD and have built a prototype tool to support what can be called a hypertext approach to documenting and browsing software. As many of the issues are related to what's been talked about, I thought people might be interested in it...

First of all, the major interest of our work was documentation support for software maintenance, especially as a means of supporting collaboration between team members. Maintenance programmers spend much of their time understanding other people's code and it is for them documentation should be produced. However, as anybody who has ever tried to maintain code knows, a lot of code has documentation which is inadequate or non-existant. A major concern was therefore that the tool should support the process of redocumentation during maintenance. There's no reason, however, why the system can't be used during development.

We use the term "rationale" to illustrate the fact that a wide variety of information should be able to be recorded, not just traditional stuff such as what this function is used for. For example, this might be comments from an inspection. A number of papers have been published describing the work: "Group Support for the Recording and Sharing of Maintenance Rationale", Robert Lougher and Tom Rodden, IEE/BCS Software Engineering Journal, November, 1993. "Supporting Long-term Collaboration in Software Maintenance", Robert Lougher and Tom Rodden, presented at the Conference on Organisational Computing Systems (COOCS'93), Nov. 1-5, Milpitas, CA, ACM.

Here's the abstract from the first paper. "This paper presents a system which supports maintenance by allowing members of a maintenance team to easily construct and structure a shared pool of maintenance information. Through the provision of a range of unconstrained documentation facilities, documentation can be constructed incrementally by any member of the maintenance team. The general model adopted exploits hypertext technology to allow documentation to be integrated into the source code, by linking comments to the appropriate source component. Structuring facilities are provided which allow the source code and documentation to be seen at different levels of abstraction, in addition to a comment typing mechanism which allows different types of information to be explicitly represented."

The system is essentially an interactive program editor in which hypertext links are displayed by highlighting. Different comment types are displayed using different styles of highlighting, i.e, foreground/background colour, bold etc. Documentation is created by annotation, i.e., the use of a variable might be explained by linking a "Variable Usage" comment to it.

The editor in addition supports a wide range of facilites. These include cross-referencing (variables, functions etc.), folding, whereby

sections of code can be grouped and folded away, display filters and graphical overviews of the network. Filtering allows complex queries to be formulated, i.e. show me all the "Bug Reports" created by Fred since last week. The overviews include views of the fold structure and graphs of the hypertext net. The system supports multiple files.

Here's the bit relevant to the current discussion. The documentation is stored in the source code files via a markup language. Rather than use an existing standard, I developed my own. As has been mentioned in previous posts, the syntax of SGML is complex. I think one of the reasons literate programming hasn't been as successful as it could have is that the print-out may be pretty, the code is however, less readable than it was before. This would be even worse with SGML. My markup syntax is simple. It has been designed to alter the appearance of the source code as little as possible. All information is commented out according to the convention of the language being used. This means the source code is still compilable (i.e. no tangling) and it may still be edited in a normal editor.

A previous poster mentioned the problem of linking definitions and uses together. I take the same approach and allow the user to indicate which should be linked. This was to keep the system language independent. The markup language's syntax is simple enough however, to allow the source code to be automatically marked up where suitable cross-referencing tools are available (i.e. cxref for C). The papers are also available as technical reports from the department. I'm currently writing my thesis so I'd welcome any comments.

From: **Andrew Dobrowolski**
Date: 22 Jan 1994

Jeffery McArthur writes: Literate programming is supposed to make programs easier to understand. Can anyone honestly say that the following is easy to understand?

```
: <chunk name="a sample chunk" file="sample.c" index="mainindex">
: #include <stdio.h>
:
: <d>main</d>() {
:    int <d>i</d>;
:    for (i=0;i<10;i++)
:       <cr ref="another chunk">
: }
: </chunk>
```

Besides being ugly it is not even good ANSI C. There is no reason that < and > should be used for tagging. SGML allows you to use other characters. C/C++ programs generally only use the characters in the range [1-127]. This leaves 128 characters in that could be used for tag delimters. Pick any two and use them instead. Personally, I prefer literate programming tools that understand and can pretty print the language.

I agree that editing SGML with an ascii editor is not for the faint of heart. To ephasise your point one more time... the less than sign in the "for" loop of the above code fragment would have to be represented by the ascii character sequence "<". Fortunately with a customizable SGML editor and a good design for the literate programming application the code would not look anything like that. It would look more like an average page from "TeX: The Program". And the less than sign would look like a less than sign.

From: **C. M. Sperberg-McQueen**
Date: 22 Jan 1994

Jeffery McArthur writes: (among other things): Consider the discussions in comp.text.sgml about ambiguous DTD's. Any language that allows you to create a "grammer", a.k.a DTD, that is ambiguous has serious problems.

Hmm. Does this mean we can all forget about regular expressions, Backus Normal Form, Generalized Phrase Structure Grammar, and every other grammar formalism known to linguistics or computer science? They all allow the construction of ambiguous grammars, or did last time I read a book on parsing theory.

Unfortunately, SGML appears to be here to stay. Nothing else seems to match what SGML does. I prefer to work with a small subset of SGML; not the entire hideous beast.

Fair enough; many of us who work some or all the time with tools which are not SGML-aware do use subsets. I understand your frustration with some aspects of SGML qua formal language; you even mention one of my pet hobby horses (record-end treatment). But the statement you make with which I agree most strongly is "Nothing else seems to match what SGML does."

Nothing else even comes close. For serious work with textual information, SGML is head, shoulders, chest, belly, and thighs above all of its competitors. But perhaps a discussion of sgml and its strengths and weaknesses belongs on comp.text.sgml, not on comp.programming.literate -- sorry for the digression, literati!

From: **Jeffrey McArthur**
Date: 22 Jan 1994

I agree that editing SGML with an ascii editor is not for the faint of heart. To ephasise your point one more time... the less than sign in the "for" loop of the above code fragment would have to be represented by the ascii character sequence "<".

Part of my philosophical disagreement is with using < and > for something they were not intended. < and > have a defined meaning in most computer languages: C, C++, Algol, Pascal, Modula-2, Ada, and so on. Forcing the less-than to be represented by "<" is counter-intuitive. This is counter to the whole concept of literate programming. If you show the SGML file to someone who has never seen a literate program they will not be able to understand it at all.

With one of the WEB decendants that is not quite as bad a problem. The WEB programs were designed with formatting computer programs in mind. SGML was designed for tagging text. In convensional text less than and greater than are quite infrequent. In computer programming they are very common. Let me illistrate my point. Consider the following example:

```
8chunk name="a sample chunk" file="sample.c" index="mainindex"9
#include 8stdio.h9

8d9main8/d9() {
   int 8d9i8/d9;
   for (i=0;i<10;i++)
      8cr ref="another chunk"9
}
8/chunk9
```

This is the same example but using 8 instead of < and 9 instead of >. This code is unreadable. You could, in theory, set up your sgml parser to accept this. 8, and 9 occur relatively infrequently and when they do you could type &eight; and &nine;. Do you see how ludicrous this is?

On another point, most WEB tools are publicly available. They are well documented and easy to use. It is not that difficult to write you own. This is not true of SGML. SGML was designed by a lawyer. And it shows! The more I deal with SGML the more I hate it. It is a good idea gone very, very bad. I think SGML has the same problems as Algol-68. The full standard has everything but the kitchen sink included.

Here is the final telling point. Write out a BNF grammer of the DTD you are using. (Unless you have defined an ambigous grammer this should be possible.) Now obtain a BNF grammer of your favorite programming language. Compare the size and complexity. In most cases the DTD will have a more complex grammer. That says alot about the complexity of SGML. (Of course you could do extreme examples like comparing Oberon to something like CALS.)

From: **Mark Carroll**
Date: 23 Jan 1994

Jeffery McArthur writes: Literate programming is supposed to make programs easier to understand. Can anyone honestly say that the following is easy to understand?

```
: <chunk name="a sample chunk" file="sample.c" index="mainindex">
: #include <stdio.h>
:
: <d>main</d>() {
:    int <d>i</d>;
:    for (i=0;i<10;i++)
:       <cr ref="another chunk">
: }
: </chunk>
```

I rather disagree here... ne of the biggest problems that I've found with literate programs is that the source code, in non-formatted form is an incredible mess for a human reader. The reason that I decided to implement the tool is precisely because I don't like reading literate code online. When I view it in pre-formatted form, there's so much extra noise that it's difficult to read. After formatting, I can only view it through a dvi viewer. What this ends up meaning, in my experience, is that you've got a computer system sitting in front of you, and you're completely unable to take advantage of the abilities of that computer to help you understand the code that you're reading. You can print the program on paper, or you can view it on virtual paper. I wanted to be able to experiment with a tool that would allow me to see my program in a useful form online, and to take some advantage from the fact that I was using a computer to view it.

Given that goal, and given that I'm not at all sure that this is really going to be advantageous, I don't want to end up wasting too much time. So I'm working with what I have available. Realistically, what I have available to work with, without implementing a complete new hypertext system is either texinfo, or HTML. Given that choice, I did some experimenting, and decided that HTML is preferable.

So I'm working within HTML syntax. Which is, admittedly rather ugly. But is it *really* all that much more ugly than a program marked up with TeX? Most of what makes the program above ugly is going to have to be present in *some* form. For my purposes, I cannot write a language specific tool (because I need to be able to work with perl, icon, C, C++, Eiffel, and perhaps Sather). So I need to provide *some* way of allowing the programmer to provide markup him/herself within the program to generate use/def links. Is the following *really* so much less ugly?

```
<<A Sample Chunk>>=
#include <stdio.h>

[[main]]() {
   int <d>i</d>;
   for (i=0;i<10;i++)
      <<another chunk>>
}
@
```

Personally, I don't really think so. And especially, since using abbreviations (and a minor syntactic change that I've done), it could be written:

```
<chunk name="a sample chunk" file="sample.c" index="mainindex">
#include <stdio.h>

<d/main/() {
   int <d>i</d>;
   for (i=0;i<10;i++)
      <cr/another chunk/
}
</chunk>
```

Besides being ugly it is not even good ANSI C.

Sorry. I was just trying to throw together a quick example. It wasn't intended to be a paragon of fine programming!

There is no reason that < and > should be used for tagging. SGML allows you to use other characters. C/C++ programs generally only use the characters in the range [1-127]. This leaves 128 characters in that could be used for tag delimters. Pick any two and use them instead.

Well, there is a reason: a lot of the HTML parsers that exist are rather sensitive about syntax. In order to be fairly certain that they aren't going to choke on my, I'm trying to avoid *anything* that doesn't commonly occur in HTML documents. I've yet to see any HTML using anything as markup other than <>, so I'm sticking with the standard.

Personally, I prefer literate programming tools that understand and can pretty print the language.

I've already explained why I'm avoiding language specific features. But there's really no reason why the code from chunks in this HTML syntax can't be piped through a formatter to do prettyprinting and definition marking automatically.

From: **C. M. Sperberg-McQueen**
Date: 23 Jan 1994

If it had just been one posting, I would have ignored it, but two postings claiming that in an SGML-based literate programming system the line

```
      for (i=0;i<10;i++)
```

would have to be represented

```
      for (i=0;i&amplt;10;i++)
```

are too many for me. Even in the reference concrete syntax of SGML, the less-than sign in this line will not be mis-recognized as a

start-tag delimiter, because it is not followed by a letter. If a variable name is substituted, the line can still be rewritten without resorting to an entity reference:

```
for (i = startpoint; i < endpoint; i++)
```

will cause no problems for any conforming parser. Some people will say white space, properly used, actually makes it more legible. If you don't like SGML for literate programming, don't use it. But before you pontificate on its syntax in public, perhaps you could be persuaded to learn a bit more about that syntax. The lawyer is several steps ahead of you.

From: **Tony Coates**
Date: 24 Jan 1994

There have been a number of posts recently discussing the use of SGML/HTML mark-up techniques in constructing literate programming tools. However, it seems to me that some recent posts for and against are arguing at cross-purposes. Particularly, I refer to the argument that ASCII SGML/HTML sources are difficult to read, at least compared to existing WEB sources. I won't deny that this point may be true; I would question though whether it is relevant.

Virtually all the current WEB tools are batch oriented, taking ASCII sources and producing TeX output, which can later be viewed. Batch-mode operation is common with new types of programs, avoiding the complications of real-time interface handling, but personally I look forward to the day when the literate programming tools will be interactive, actively formatting the code as it is input, much like a modern word processor. Much as I like working with LaTeX for my sources, I don't see a lot being done by way of creating any kind of "real-time" TeX, which would work in other than batch mode. On the other hand, at least with the HTML extensions to SGML, hypertext offers interactive browsing of code, and in principle the marked-up code should be able to be formatted on the fly as required, I believe.

So it seems to me that on that basis, SGML/HTML markups may offer useful advantages in generating a way of coding the literate programs so as to be able to exploit the features of HTML/SGML. Still, it is clear that if users were to be editing ASCII SGML sources, the experience might be off-putting.

However, since SGML sources are meant to be for the machine, and in principle SGML editors which shield the user from such source code are suppose to be possible or even available, is the better question not that of how well the required mark-ups for literate programming could be added using a proper SGML editor, and what this would look like and feel like for the end-user? I have no experience with such editors, but would be interested to hear from someone who does. I sometimes imagine opening up a WYSIWYG editor, filled with documentation and code, and then highlighting a paragraph of code, selecting "Code" from the menu, and then defining the name of this code block, with the editor providing the appropriate formatting. If SGML editors could offer this, I for one would consider that building literate programming systems using SGML mark-ups as the base could be a worthwhile step forward.

Anyway, this is long, but I did feel that too much was made of what SGML sources would look like, when this did not seem to me to be the question.

From: **Eric Skinner**
Date: 24 Jan 1994

Steve Heaney writes: It occurs to me that SGML (the Standard Generalised Markup Language) provides many possibilities for supporting a software development environment. Because SGML provides the means of defining and verifying the content and structure of a "document" it offers, for example, the potential for coupling the documentation produced by software process (requirements, spec, design docs) with program source and beyond to maintenance manuals.

[...] Maybe someone has tried some of this already. I would be very interested if anyone has DTD's or DTD fragments for program specification or design, especially if these have been coupled with tools for generating or validating the resultant source code.

Exoterica's OmniMark product contains our commercial SGML parser, the SGML Kernel. The Kernel is built using an approach similar to what you described in your posting. The Kernel is in fact coded in an SGML markup language, combining engineering documentation with source code in a language close to C. The coding language is extended using SGML-defined constructs to provide a notation specifically useful for writing SGML parsers. It's a Web-like approach, but more powerful as code and documentation are more closely integrated. Our OmniMark product is then used to read this SGML document and generate C code, or to generate various types of documentation. OmniMark can also perform a variety of consistency checks.

We also have an internal highly-detailed marked up version of ISO8879 which we use for many purposes, including the generation of our online hypertext version of the standard, part of the "Compleat SGML". The markup in this document is rich enough that it allows OmniMark to generate some C code for the Kernel directly from the markup of ISO8879's productions. The advantages we found in this type of development were many, including greater ease of coding, documentation, and faster development of high-quality code.

From: **Glenn Vanderburg**
Date: 24 Jan 1994

Steve Heaney writes: Literate programming is supposed to make programs easier to understand. Can anyone honestly say that the following is easy to understand? [Silly example of C code marked up with SGML tags deleted] Personally, I prefer literate programming tools that understand and can pretty print the language.

So do I. Which is why I am very interested in a literate programming tool based on SGML. When I read your posting, I spotted a couple of assumptions you made which are bogus. Here are my contrary views:

1. Most users of decent SGML applications will not need to read the SGML source. I know SGML, and I actually *like* the choice of `<' and `>' as tag delimiters. But I have no desire to work with the SGML source in a text editor, and I consider any SGML browser/editor which requires that to be crippled. I want a literate programming tool which will permit me to work online with something that resembles the formatted, printed result of the WEAVE process. I think that SGML makes that an easier task.

2. Everything in an SGML document does not have to be marked up with SGML tags for an SGML processor to understand its structure. Learn about SGML's notion of `notations'. C is a perfect example of a data notation which is not defined in terms of SGML. A notation parser which knows C will not need SGML tags to recognized declarations. That information is already present in the C notation, and the C parser can cooperate with the SGML parser, informing it of the structure. If I were designing an SGML literate programming application, your example would be marked up like this:

```
<chunk name="a sample chunk">
#include <stdio.h>

main() {
   int i;
   for (i=0;i<10;i++)
      <cr ref="another chunk">
}
</chunk>
```

That's much more reasonable, and as I've already said, users of the application should see this presented in an even nicer way. I can't criticize you too much for making those false assumptions, because SGML applications that really work this way are rare and expensive. And you do make some good points. SGML is feature-laden, and some of the features (like data tags, short references, and some of the other minimization features) are useful primarily to people who will be working directly with the SGML encoding. But I think that soon, applications will be available which hide that from users.

From: **Kayvan Sylvan**
Date: 24 Jan 1994

Tony Coates writes: There have been a number of posts recently discussing the use of SGML/HTML mark-up techniques in constructing Literate Programming Tools. However, it seems to me that some recent posts for and against are arguing at cross-purposes. Particularly, I refer to the argument that ASCII SGML/HTML sources are difficult to read, at least compared to existing WEB sources. I won't deny that this point may be true; I would question though whether it is relevant. [...]

Exactly. It seems to me analogous to arguing that one should not write computer programs because the underlying sequences of ones and zeros are hard to understand. The SGML/HTML markup is for use by tools that would hide the ugly details from the users. I program in WEB because I want some protection from the ugly details of the programming language ;-). I would welcome a chance to program in an interactive HTML editor that would protect me from the ugly details of WEB.

From: **Robert Lougher**
Date: 25 Jan 1994

Tony Coates writes: There have been a number of posts recently discussing the use of SGML/HTML mark-up techniques in constructing literate programming tools.

Yes, including mine which seemed to fall on deaf ears! I believe I made a number of interesting points which haven't been picked up on. Maybe I didn't make my post as relevant to the current discussion as I could have... If you remember :-), I mentioned the system I've developed for my PhD which allows programs to be documented and browsed using hypertext (I also put in rather a lot of blurb - I am writing my thesis!).

Virtually all the current WEB tools are batch oriented, taking ASCII sources and producing TeX output, which can later be viewed. Batch-mode operation is common with new types of programs, avoiding the complications of real-time interface handling, but personally I look forward to the day when the LitProg tools will be interactive ... ... with the HTML extensions to SGML, hypertext offers interactive browsing of code, and in principle the marked-up code should be able to be formatted on the fly as required, I believe.

In a nutshell, my system allows a program to be (re-)documented by attaching annotations to the source code using hypertext links. These can be attached to identifiers, program lines and code segments. Code segments are specified using folds. These allow a number of program lines to be grouped together and folded away. Annotations can be made on annotations and more than one link can be made to the same annotation (i.e. a graph structure). Hyper-links can also be used to browse the code, i.e. links between variable/function use and definition. This is all done in a WYSIWIG manner with windows. The code may be edited, folded and annotated interactively, and graphical views can be generated of the hypertext network. Unfortunately, to make the system language independent no formatting of the source code is performed.

So it seems to me that on that basis, SGML/HTML markups may offer useful advantages in generating a way of coding the literate programs so as to be able to exploit the featrues of HTML/SGML. Still, it is clear that if users were to be editing ASCII SGML sources, the experience might be off-putting.

I agree.

However, since SGML sources are meant to be for the machine, and in principle SGML editors which shield the user from such source code are suppose to be possible or even available, is the better question not that of how well the required mark-ups for literate programming could be added using a proper SGML editor, and what this would look like and feel like for the end-user?

When developing my system, most of the time was taken up with the development of the editor. This included issues of how should the link anchors be presented, the traversal of links, the creation of annotations, the support for browsing etc. A MAJOR consideration however, was the underlying storage. It is far easier to save the code+documentation in some unreadable format (as a word processor). This however, locks the source code into the system as it can only be modified using the editor (which may not always be available).

I took the mark-up approach instead. However, rather than use a standard markup, I developed my own. This was because: a) it was 2.5 years ago b) you can develop a markup which is suited (i.e. readable) to marking up programs. HTML is angled towards standard text (i.e. lists, paragraphs etc.). I believe a markup for programs should alter the appearance of the source code as little as possible (i.e. indentation). It should be possible to edit the program occasionally in a conventional editor using the markup, or in an interactive WYSIWYG manner using the hypertext system. While my markup isn't perfect, it goes a long way to achieving that.

From: **Lee Wittenberg**
Date: 25 Jan 1994

Jeffry McArthur writes: Literate programming is supposed to make programs easier to understand. Can anyone honestly say that the following is easy to understand?

```
: <chunk name="a sample chunk" file="sample.c" index="mainindex">
: #include <stdio.h>
:
: <d>main</d>() {
:    int <d>i</d>;
:    for (i=0;i<10;i++)
:       <cr ref="another chunk">
: }
: </chunk>
```

I've mentioned before and feel compelled to mention again that the purpose of the web (i.e. markup) is to generate readable versions rather than to be read directly. Tangle generates a file to be read by a compiler, and weave generates a document to be read by humans. Presumably, other filters could generate stuff for other purposes. I should probably stop ranting about this, but I feel it is important.

Besides being ugly it is not even good ANSI C. There is no reason that < and > should be used for tagging. SGML allows you to use other characters. C/C++ programs generally only use the characters in the range [1-127]. This leaves 128 characters in that could be used for tag delimters. Pick any two and use them instead.

Not knowing SGML, I had assumed that < and > were traditionally used (or required) for tags. If Jeffrey is right about SGML allowing other characters, then I agree with him about choosing something that doesn't interfere with programming languages. I believe that's why DEK chose @< and @>.

Personally, I prefer literate programming tools that understand and can pretty print the language.

Me too, but I'm getting to where the free-form style of the document in noweb is more important to me than pretty-printing. On the other hand, I still dream of a filter (still notes on paper) that will pretty-print in noweb.

From: **Lee Wittenberg**
Date: 25 Jan 1994

Robert Lougher writes: In a nutshell, my system allows a program to be (re-)documented by attaching annotations to the source code using hypertext links. These can be attached to identifiers, program lines and code segments. Code segments are specified using folds. These allow a number of program lines to be grouped together and folded away. Annotations can be made on annotations and more than one link can be made to the same annotation (i.e. a graph structure). Hyper-links can also be used to browse the code, i.e. links between variable/function use and definition. This is all done in a WYSIWIG manner with windows. The code may be edited, folded and annotated interactively, and graphical views can be generated of the hypertext network. Unfortunately, to make the system language independent no formatting of the source code is performed.

This all sounds very nice, but can it print out a human-readable document as well? The reason I ask is that I spent much of the last year struggling with visual programming systems where the only way to find out which bit of code was attached to an "object" (the system can best be described as widget-oriented rather than object-oriented, but had delusions of grandeur) was to click on the object with the mouse. Code maintenance became next to impossible, because there was no way, short of clicking on every object, to find out where variables and functions (and objects) were accessed.

We solved the problem by building a "decompiler," that produced a text representation of the complete program (in a home-grown mini-language), but this is not possible in all visual programming systems. The text representation was also useful when we wanted to change all the blue objects to green, or some such (we're also working on a compiler to reverse the process, but the VP system we're using is not very helpful in that regard). In any event, we can use all the lovely Unix text tools (diff, grep, etc.) to find out information about our visual programs.

In any event, may I request that you include in your system some way to a) produce human-readable (typeset in some way) documents from your hypertext base, and b) produce some text version that can be analyzed by the tools we all know and love (so we don't have to go out and buy more tools :-).

From: **Lee Wittenberg**
Date: 25 Janr 1994

Robert Lougher says (among other things I happen to agree with): I think one of the reasons literate programming hasn't been as successful as it could have is that the print-out may be pretty, the code is however, less readable than it was before. This would be even worse with SGML. My markup syntax is simple. It has been designed to alter the appearance of the source code as little as possible. All information is commented out according to the convention of the language being used. This means the source code is still compilable (i.e. no tangling) and it may still be edited in a normal editor.

I disagree with this. I find literate code more readable. Remember that what is traditionally source is designed to be filtered before being read. Through tangle, if the reader is a compiler; through weave, if the reader is human. In addition, I find that even without explanations, code written as code chunks is much easier to digest, particularly when it's someone else's code. I'm experimenting with this with my CS1 students (more on this in a later message), and they seem to find untangled webs easier to understand than the complete ("tangled") program, primarily because the chunks remain small and complete in themselves.

From: **John Ramsdell**
Date: 15 Apr 1994

SGML enthusiasts claim that the logical structure of a document can be captured using SGML is a fashion that allows layout decisions to be made at a later date. In particular, with a carefully designed Document Type Definition (DTD), one could encode literate programs which can be displayed as printed text and hypertext.

For literate programming, one would start with a DTD that encodes technical documents, presumably including facilities for tables and equations. The DTD would be modified so as to allow the inclusion of code scraps and their references. Documents using the literate programming DTD could be translated directly into LaTeX or HTML, or it could be translated into input for your favorite WEB program. For example, for C programs, the target of your translation could be CWEB, FWEB, or Nuweb. Has any one thought about using SGML for literate programming by producing a DTD tuned just for this purpose?

From: **Tommy McGuire**
Date: 15 Apr 1994

John Ramsdell writes: In particular, with a carefully designed Document Type Definition (DTD), one could encode literate programs which can be displayed as printed text and hypertext.

As an aside, this is not as simple as it appears. One of the recent flame wars topics on the SGML mailing list has concerned the problem of getting both printed docs and hypertext from the same source. Doing this apparently requires compromises in at least one of the two products.

For literate programming, one would start with a DTD that encodes technical documents, presumably including facilities for tables and equations. The DTD would be modified so as to allow the inclusion of code scraps and their references. Documents using the literate programming DTD could be translated directly into LaTeX or HTML, or it could be translated into input for your favorite WEB program. For example, for C programs, the target of your translation could be CWEB, FWEB, or Nuweb. Has any one thought about using SGML for literate programming by producing a DTD tuned just for this purpose?

The QWERTZ DTD and application includes a simplified literate programming system. QWERTZ (also known as <mumble>FORMAT<mumble> at some ftp sites) is nearly a direct translation of LaTeX tags into a SGML DTD. It provides nearly direct translation into LaTeX as well as somewhat indirect translation into troff and HTML. (So I hear. I havn't actually seen the HTML output.)

The literate programming environment involves a "code" tag. The code scraps are concatenated in the order that they appear in the source when producing a machine-usable file and are formatted similarly to nuweb in a LaTeX file. This works pretty well for things that are not structurally decomposed, such as the QWERTZ DTD itself. Also, it wouldn't be too hard to add an attribute or three to the code element for a scrap name, enabling reordering and embedding of scraps. I havn't tried this since I am rather fond of getting correct line numbers along with error messages from compilers, for example, and I havn't figured out a way to do that. Disclaimer: I havn't done much with SGML or HTML. I just like the flame wars on the SGML mailing list.

Index for function definitions

From: **Joachim Ziegler**
Date: 21 Jan 1994

I use CWEB for "webbing" C++ programs. I would like to ask other users if they miss the same feature as I do: At the end of each section, you get the references "See also Section blah blah ..." and "This code is used in Section blah blah ...". But what I really miss is that for each function call in the C++ portion of a section there should be a reference to where the code of this function is really defined, not declared, that means something like this: "For the definition of function foo() see Section X.". Looking at the index table at the end of the tex-output may make you search through a lot of sections until you find the code of a specific function. Any opinions?

From: **Joachim Schrod**
Date: 22 Jan 1994

Joachim Ziegler writes: I use CWEB for "webbing" C++ programs. I would like to ask other users if they miss the same feature as I do: At the end of each section, you get the references "See also Section blah blah ..." and "This code is used in Section blah blah ...". But what I really miss is that for each function call in the C++ portion of a section there should be a reference to where the code of this function is really defined, not declared, that means something like this: "For the definition of function foo() see Section X.". Looking at the index table at the end of the tex-output may make you search through a lot of sections until you find the code of a specific function. Any opinions?

CWEB does _not_ contain a C++ parser, so it does not know the difference between declaration and definition. Actually, due to the syntactic ambiguity of C++ this isn't easy to add, either. Even then, a pure syntactic recognition is not enough because one has to handle overloading and method redefinitions. (You remember, in every OO language you *cannot* be sure in general what function is used at a given place.) In addition, such a process would not crossref to other modules (that are usually implemented as other translation units in C++), you are stuck within one file.

And, since this came up in the past in this context: Personally, I find the idea of putting more than one module in a *WEB source and extract them in different C++ files the most horrible thing I've ever heard of. That goes straight against everything we need and want in terms of software engineering. (Most of all: separation of concerns, information hiding, introduction of abstraction levels, encapsulation; in short, the whole concept of modularization blows away then.) I would rather abandon literate programming than abandon the things we learned about the construction of software systems the last twenty years. Happily, it isn't necessary...

From: **Zdenek Wagner**
Date: 24 Jan 1994

Joachim Ziegler writes: I use CWEB for "webbing" C++ programs. I would like to ask other users if they miss the same feature as I do: At the end of each section, you get the references "See also Section blah blah ..." and "This code is used in Section blah blah ...". But what I really miss is that for each function call in the C++ portion of a section there should be a reference to where the code of this function is really defined, not declared, that means something like this: "For the definition of function foo() see Section X.". Looking at the index table at the end of the tex-output may make you search through a lot of sections until you find the code of a specific function. Any opinions?

I do not see such problem. Remember that CWEB (as any WEB) enables you to place any chunk anywhere which seems logical and convenient for you and ctangle rearranges them so that the compiler understands the program. If you have problems finding something in *your* programs, it only reveals that you still think as your compiler. Try to rearrange your web file in the way you best understand. This is one of the big advantages of literate programming.

From: **Bart Childs**
Date: 26 Jan 1994

Joachim Zeigler's post about indexing was responded to by Joachim Schrod. I wish to add another comment to this. There is a WEB command (@! in the original, @_ in FWEB 1.40, and I don't have a CWEB manual handy) that will underline an index entry. I am not keen on it all being automatic and find this extra burden on the author to be quite small. I believe the mindset of literate programming should be to add these commands, supplement the default index by user supplied entries, ... I think this quote is appropriate: "What is written without effort is in general read without pleasure." -- Samuel Johnson The word `pleasure' could be replaced by `ease and pleasure'.

From: **Marc van Leeuwen**
Date: 26 Jan 1994

Joachim Schrod writes: CWEB does _not_ contain a C++ parser, so it does not know the difference between declaration and definition. Actually, due to the syntactic ambiguity of C++ this isn't easy to add, either.

I have no experience with CWEB used for C++, but I think I disagree with this statement. Certainly CWEAVE contains a parser for C/C++, albeit an imprecise one that will accept more than a compiler will, and I think it in fact does distinguish between function declarations and definitions: consecutive function declarations will be simply set on consecutive lines, but in case of consecutive function definitions some white space is inserted in between (at least this is what happens in my version of CWEB). This is just to indicate that the distinction is not too difficult to make (after all, is it not just looking for an opening brace after the closing parenthesis of the parameter list?). It is true though that CWEB/WEB use of the term "module" in the cited text; I prefer to speak only at present does not make use of the distinction: in both cases the function name will appear underlined in the index. If you want to use the information in the typeset document in sections where the function is used, you have a problem however, since parsing takes place in the same pass of CWEB where the TeX output is produced so any attempt to make this work in the current setup of CWEB would only work if the function is defined (in the CWEB source) before it is called.

*Even then, a pure syntactic recognition is not enough because one has to handle overloading and method redefinitions. (You remember, in every OO language you *cannot* be sure in general what function is used at a given place.) In addition, such a process would not crossref to other modules (that are usually implemented as other translation units in C++), you are stuck within one file.*

I think this is a much more convincing reason why it really would be a big problem. If your program is compiled from say 50 different units, each one with its own CWEB document, then many functions will appear in many indexes as "used once or more times, but nowhere defined" (because the declaration is contained in an included header file which CWEAVE does not see). [As a side note, I think a reference to the relevant #include statement from the index would not be desirable either, because the index would then be swamped with functions being defined (i.e., declared) but not used.] In any case CWEB provides no help in such situations to find the defining file; it might be useful to augment CWEB with another program that manages such global interconnections. By the way, the term "module" is used above in a way that is non-standard in WEB context (yes the word already had many meanings before WEB came around); I prefer to call that what is processed by a single run of CTANGLE or CWEAVE a single CWEB source (even if residing in more than one input file) and its result on paper a CWEB document.

*And, since this came up in the past in this context: Personally, I find the idea of putting more than one module in a *WEB source and extract them in different C++ files the most horrible thing I've ever heard of. That goes straight against everything we need and want in terms of software engineering. (Most of all: separation of concerns, information hiding, introduction of abstraction levels, encapsulation; in short, the whole concept of modularization blows away then.) I would rather abandon literate programming than abandon the things we learned about the construction of software systems the last twenty years. Happily, it isn't necessary...*

I suppose extracting means using the multiple file output to produce several compilation units from one CWEB source. I agree that this is a bad idea, although it is an entertaining thought to consider using multiple input files as well, each of which roughly corresponds to a translation unit; an advantage would be that CWEAVE could produce one global index. It is easy to dream up numerous disadvantages as well though, for instance the fact that keeping module names unique becomes a serious problem and that "make" would become super-conservative, recompiling the whole thing every time CTANGLE is used to update one translation unit (this is already a problem if you use multiple output files in a decent way, i.e., to generate header files, and you are honest to "make" about dependencies on header files).

From: **Norman Ramsey**
Date: 26 Jan 1994

There is a WEB command (@! in the original, @_ in FWEB 1.40, and I don't have a CWEB manual handy) that will underline an index entry. I am not keen on it all being automatic and find this extra burden on the author to be quite small.

This scheme is fine for archival code (write once, read many), but it's a problem for code that is edited frequently, because the intercalated markup distracts from the code. IMHO, the proper solution is to have an editor that hides the markup, with "display hidden characters" available for those rare times you need to see it. Until a righteous emacs hacker shows us how to do this, I'll continue to crusade against markup intercalated between program elements. My metacomment is that not only will many literate programs never be published, many will be read only by people who are about to modify them. In this case, having the source and document correspond wins.

Reengineering and literate programming

From: **Jim Crigler**
Date: 25 Jan 1994

Has there been any work known to you of using literate programming as part of a reengineering effort, specifically w.r.t. documenting the existing system. (Yes I am following the thread on SGML and literate programming.)

From: **Tony Coates**
Date: 25 Jan 1994

Jim Crigler writes: Has there been any work known to you of using literate programming as part of a reengineering effort, specifically w.r.t. documenting the existing system. (Yes I am following the thread on SGML and literate programming.)

The closest that I personally am aware of is Ross Williams' first FunnelWeb project, where he analysed and diagnosed problems in a postscript program by converting it to a literate program, adding explanations of each code section as he understood it. I suppose that this is something along the lines of what you are interested in. Ross certainly felt that using FunnelWeb in this way helped him find the problems long before other methods.

From: **Dave Thompson**
Date: 25 Jan 1994

The closest that I personally am aware of is Ross Williams' first FunnelWeb project, where he analysed and diagnosed problems in a postscript program by converting it to a literate program, adding explanations of each code section as he understood it.

I've been working (off and on) on a re-engineering of a minimization algorithm. This is my exercise to learn something about literate programming from a pracitical (rather than theoretical) perspective. I'll keep track of my results, thoughts, and mistakes. Maybe there will be some interested in the outcome of my exercise?

From: **Lee Wittenberg**
Date: 25 Jan 1994

Jim Crigler asks: Has there been any work known to you of using literate programming as part of a reengineering effort, specifically w.r.t. documenting the existing system. (Yes I am following the thread on SGML and literate programming.)

Yes, Carl Gregory of Tipton Cole+Co. has been doing just that with noweb and PAL (Paradox Application language). You might want to write him direct at gregory@utkcs2.cs.utk.edu (he doesn't subscribe to this list, alas). I also rewrote a non-literate library in a CWEB version (available as eventlib under pub/leew/samples.LP at bart.kean.edu), but that was mostly a personal experiment (both with respect to the literate program and eventlib itself), not a system under real maintenance, so I don't guess that's what you're looking for. Then again, maybe it is. Let me know.

From: **Thorbjoern Andersen**
Date: 26 Jan 1994

Tony Coates writes: The closest that I personally am aware of is Ross Williams' first FunnelWeb project, where he analysed and diagnosed problems in a postscript program by converting it to a literate program, adding explanations of each code section as he understood it. I suppose that this is something along the lines of what you are interested in. Ross certainly felt that using FunnelWeb in this way helped him find the problems long before other methods.

The PostScript programs I have seen were rather cryptic, and deciphering them were very much like disassembling a machine code program which basically goes in the same lines. As I soon may be forced to write some assembly code again, but wants to limit the lack of readability I wish to do so in *WEB , especially because it may be handed out. Does anybody have experience with this?

From: **Marcus Speh**
Date: 30 Jan 1994

The following describes a "reengineering", sort of - subsequent refinements of parallel code using litprog techniques. A highly readable account, IMO. Available from pobox.cscs.ch via anon FTP in Directory pub/SeRD/TechReports. Haven't checked whether it is in the LitProg library, though.

```
@TECHREPORT{,
        NUMBER {IAM 91-018},
        NOTE = {Filed: yes, PSFile: yes},
        AUTHOR = {Karsten M. Decker}
        TITLE = { A Methodology for the Design and Implementation of Efficient
                  Algorithms for Scalable Parallel Architectures}
        INSTITUTION = {Institut f\"ur Informatik},
        YEAR = {1991},
        TYPE = {Technical Report},
        ADDRESS = {University Bern, Switzerland},
        MONTH = dec,
        ABSTRACT = { Parallel programs exploiting as much as possible of the
                     nominal performance of distributed memory parallel
                     processor systems not only require efficient
                     implementations but in particular careful algorithm
                     design. In this paper we propose a design and
                     implementation methodology for parallel algorithms
                     incorporating the concept of literate programming and
                     report on the experience gained in the development of a
                     prototype library. The investigation is part of the
                     SPADE project which aims at the development of an
                     integrated program and application development
                     environment for scientific applications on parallel
                     architectures with distributed memory.
                   }
}
```

Usage of multiple-output files facilities

From: **Lee Wittenberg**
Date: 25 Jan 1994

Joachim Schrod writes: And, since this came up in the past in this context: Personally, I find the idea of putting more than one module

*in a *WEB source and extract them in different C++ files the most horrible thing I've ever heard of. That goes straight against everything we need and want in terms of software engineering. (Most of all: separation of concerns, information hiding, introduction of abstraction levels, encapsulation; in short, the whole concept of modularization blows away then.) I would rather abandon literate programming than abandon the things we learned about the construction of software systems the last twenty years. Happily, it isn't necessary...*

I'm not going to address the main question of whether or not putting multiple modules in a single web is a good or bad thing, but I would like to point out that the ability to extract chunks to a separate file is vital (IMHO) when defining C++ classes in CWEB.

```
@ The structure of my class webs is usually something like this:
@<Header files@>@;      // included as well as extracted
@(xxx.h@>@;
@<Member definitions@>@;

@ With the header extracted to a separate file so the class can be
used by others (using the \.{@@(}) mechanism:
@(xxx.h@>=
#ifndef _XXX_H
#define _XXX_H  // prevent re-inclusion

class xxx {
        @<Private members@>@;
protected:
        @<Protected members@>@;
public:
        @<Public members@>@;
};

#endif
```

Without the @(xxx.h@>, I wouldn't be able to make my class available in a library for other programs to use.

From: **Joachim Schrod**
Date: 25 Jan 1994

*Joachim Schrod writes: And, since this came up in the past in this context: Personally, I find the idea of putting more than one module in a *WEB source and extract them in different C++ files the most horrible thing I've ever heard of.*

Lee Wittenberg writes: but I would like to point out that the ability to extract chunks to a separate file is vital (IMHO) when defining C++ classes in CWEB.

I think it might be interesting to present my use of this facility, which is quite different from yours. To be more precise, the usage of our group that we built in the last ten years using a whole range of literate programming tools. The systems in question typically have use between 20 and 100 modules, ranging from only 100 KB to 20 MB in total. Any comments would be appreciated. Please bear with me, I want to explain a few basic premises of our working principles first:

We're in the `specification as a contract' camp. I.e., a module's specification is the document that has everything a user needs for writing his code. There must not be any need to look into the realization until binding time (where certain constraints, in particular partial implementations, run-time and space efficency, must be considered). OTOH a specification must present to an implementer everything that's needed to realize this module. In return he or she agrees to fulfill this contract. When I use a module written by another person (might have been me half a year ago ;-) I _want_ to look *only* at the specification, and I don't want to be bothered with the implementation. I want a (hopefully short and concise) description about the services this module delivers.

Second, there may be more than one realization for one specification. In fact, that's quite common for most of the systems we build. Each realization has different trade-offs. I have to select the realization at configuration time, then I need a high-level description of this representation's properties. Note, that this doesn't mean access to the code; I _trust_ my fellow co-workers. After all, this trust is regularily reinforced by code reviews. Don't you remember the great Turing Award lecture by Ritchie: "Trusting programs means trusting people"...

Third, the frequency of changes to realizations is much higher than the frequency of changes to specifications. In our group that has the consequence that before anybody of us may commit a specification change to our source repository that change must be refereed by at least one other person. (We all use proper revision management, don't we? Hey, that's what my rcs style is for... ;-) ;-) That is not needed for changes in realization, where code reviews are triggered by other things. That conforms to the viewpoint that changes in design must be considered much more carefully than changes in realizations. They also concern more modules and are not local any more!

Coming to C++: Like C this language does not have any notion about modules. We have to simulate them by translation units, where header file(s) represent a specification document and C++ file(s) represent the realization document. That means, *WEB coded files. One of the problems of C++ is that implementation details may sneak in the class declarations, but that can be avoided most often by well known C++ techniques. I.e., we don't find any use for implementation details in specification documents.

Since a realization always has a directed [sp? `gerichtet', as in directed graphs] relationship to its specification, we think it's good style to include the specification document at the start of the realization document. After all, it's the contract that this realization fulfills. For this inclusion, we use @i (or other analogue constructs) so that it gets printed in the realization document as well. OTOH, the import in a client module is done by #include since we don't want to be overwhelmed with the information about all used modules; there are simply too many.

In our method the usage for the `extract to many files' feature is mainly in the realization document. E.g., we can tangle different functions/methods to different source files, to reduce the size of the object modules. If some clients use only certain parts of a server module's protocol, smaller executables (both on disk and in memory) result if no shared libraries are available. Another example are bison/flex files. 'Though we currently switch more and more to PCCTS (Take this as an advertisement, folks! Sorry, Preston :) :) there are still old files implementing small languages where one document describing the whole language is fruitful. Well, enough musings, back to work. Hope it is of interest to some of you and might feed some discussion although it got too long...

From: **Glenn Vanderburg**
Date: 26 Jan 1994

Joachim Schrod writes: We think it's good style to include the specification document at the start of the realization document. After all, it's the contract that this realization fulfills. For this inclusion, we use @i (or other analogue constructs) so that it gets printed in the realization document as well.

This is a terrific strategy. It seems to me that, when using WEB with a language like C++, it should be possible to generate two forms of printed output for a class library: the implementation, which is a fairly traditional WEB (here's what it does, and also how it works) and an interface document, which contains the detailed interface specification, along with notes on anticipated styles of use, programming idioms which it is intended to support, etc. In general, of course, I don't advocate giving someone a WEB in place of a user's manual. But for a class library, a WEB-style listing of the interface would be terrific.

OTOH, the import in a client module is done by #include since we don't want to be overwhelmed with the information about all used modules; there are simply too many.

This is true, and it's also great that the author of the client module has a choice. Certainly, including details of all imported modules in the program would be overwhelming. But often there are one or two classes which shape the whole program; it greatly increases the readability of the program if the interfaces to those classes are presented early on.

... remember the great Turing Award lecture by Ritchie: "Trusting programs means trusting people"...

FYI ... that was Thompson's lecture.

Novice looking for recommendations

From: **Tom Epperly**
Date: 26 Jan 1994

I have read the FAQ which gives a nice list of literate programming tools. However, I am wondering if someone can give me some recommendations, so I can avoid having to check them all out. I am familiar with TeX, C, and C++, and I would like to start a big project using a literate programming language. Can you recommend the top two tools that I should consider?

I have looked at CWEB, and it seems nice except for its method of generating multiple output files. I need to generate an include file(.h) and a C++ file(.cc), and I am not sure about the pluses and minuses of using one or two WEB files. If I want to use a single file to generate both the .h and .cc files with CWEB, it seems that the code for the .h file must be named @(filename.h@> which isn't very mneumonic to me. What are the pluses and minus of a single WEB file to generate .h and .cc? I have read the CWEB manual, and I am still not sure.

From: **Andrew Mauer**
Date: 26 Jan 1994

Tom Epperly said: I am familiar with TeX, C, and C++, and I would like to start a big project using a literate programming language.

A caveat: literate programming is quite another beast. You will probably want to start out and experiment with some small projects (if this is feasible) to develop both your WRITING and PROGRAMMING style. In my experience, both needed changes to write decent webs (and, I admit, I still do not :-). I needed to get used to breaking things down into comfortable sections. At first the sections were *way too small*.... it was hard to understand (and maintain) because the code was too fragmented. (It was, however, impeccably documented, down to the last pebble. :-) After several months (still a novice by any measure), my document does not flow well (smoothly) between chunks.

Can you recommend the top two tools that I should consider?

Private email, so as not to start anything. ;-)

I have looked at CWEB, and it seems nice except for its method of generating multiple output files [...] it seems that the code for the .h file must be named @(filename.h@> which isn't very mneumonic to me.

This is probably not an important feature. It may be ugly, but after a few days of programming, you will have no problem with it. More important to consider is whether the tool will let you include other web files in your document (like @i.) and see the web file in the debugger, rather than the tangled file.

What are the pluses and minus of a single WEB file to generate .h and .cc?

(War warning...) IMHO (*very* humble O), using the same document to generate both is the way to go because it will let you keep the header defs next to the function defs (for instance), so the changes are localized if you make them. You can also weave the same informational/specification chunks into both documents. (Maybe.) However, if you're doing multiple implementations with the same interface (a la Joachim's modules), this is probably a bad way to go. [This is NOT intended as an entry into the ongoing discussion on said topic.]

From: **David Kastrup**
Date: 26 Jan 1994

Tom Epperly writes: I have read the FAQ which gives a nice list of literate programming tools. However, I am wondering if someone can give me some recommendations, so I can avoid having to check them all out. I am familiar with TeX, C, and C++, and I would like to start a big project using a literate programming language. Can you recommend the top two tools that I should consider?

I have looked at CWEB, and it seems nice except for its method of generating multiple output files. I need to generate an include file(.h) and a C++ file(.cc), and I am not sure about the pluses and minuses of using one or two WEB files. If I want to use a single file to generate both the .h and .cc files with CWEB, it seems that the code for the .h file must be named @(filename.h@> which isn't very mneumonic to me. What are the pluses and minus of a single WEB file to generate .h and .cc? I have read the CWEB manual, and I am still not sure.

In short, modularity is always a plus. If you have independently working program files, having independent web files is a plus. Independent modules are a bit harder to judge. Of course, cross-referencing of global variables can be a difficult task. Global variables are rather ugly, anyway, but global types can hardly be dismissed... Splitting a web into several sources is at least recommendable for development, even if you merge them with @i into one document.

Personally, I prefer noweb, since it allows making use of all LaTeX file splitting capabilities without thinking. It can also painlessly extract modules from whatever source combinations feasible, and has a very handy external command "cpif" which will compare and copy if the new version differs, touching no more files than necessary. In noweb, you'd write for the above

```
<<filename.h>>=
...
Or even
<<File filename.h>>=
...
```

Although you have to quote such a filename when extracting (Extracting will extract any scrap name you choose. Using the filename is

the obvious choice, except when there is only one file, where <<*>> is sufficient). Although noweb is very nice, it needs the Unix toolset (at least awk (or perl), sed, cmp, cp, some Bourne shell or clone, and a few more).

nuweb is said to be similar in flavor. If you have the Unix toolset, I'd recommend looking at least at noweb, otherwise at least at nuweb (requires C compiler). They should give you a pretty good idea of a different approach (without pretty-printing).

If you have to deliver C source at any stage of your project, noweb will give you a very good tangled source, either including the web as comments, or properly indented with regard to scrap indentation (as you formulated it) (I do not remember if both at the same time was possible or feasible). If you have to debug without #line directives (some environments force you to do that) you cannot do any better.

From: **Norman Ramsey**
Date: 26 Jan 1994

I have read the FAQ which gives a nice list of literate programming tools. However, I am wondering if someone can give me some recommendations, so I can avoid having to check them all out. I am familiar with TeX, C, and C++, and I would like to start a big project using a literate programming language. Can you recommend the top two tools that I should consider?

Poor soul, you don't know what a flame war you're about to start. Try the Levy/Knuth CWEB, and try noweb. If you substitute nuweb for noweb, only Ross will object :-)

From: **Marc van Leeuwen**
Date: 27 Jan 1994

Tom Epperly writes: I have looked at CWEB, and it seems nice except for its method of generating multiple output files. I need to generate an include file(.h) and a C++ file(.cc), and I am not sure about the pluses and minuses of using one or two WEB files. If I want to use a single file to generate both the .h and .cc files with CWEB, it seems that the code for the .h file must be named @(filename.h@> which isn't very mnemonic to me.

First of all, the code for the .h file *does not have to be* named @(filename.h@> any more than the code for the main output file has to be unnamed. The module called @(filename.h@> just serves as the root module for shipping to that file, and it can have an arbitrary tree of submodules beneath it. (Admittedly this point is less then clear from the standard CWEB documentation). Personally I usually have a module named something like @< External typedef declarations @> and others like @< Prototypes of external functions @> and @< Macros for external use @>, but you can be more elaborate is you like. Just make sure that the names clearly indicate that this stuff is going to a header file, not to the main file. By the way, I am assuming here that header files are used just to ensure type consistency across compilation units, not as a specification mechanism; for that case a good suggestion is made in a recent posting by Joachim Schrod. Since there has been some discussion on multiple output files recently, let me elaborate this a bit with some remarks that I have missed in the discussion. You can start your CWEB source with something like

```
@c
@< Include files @>
@< Local typedefs @>
@< Local prototypes @>
...

followed by a new module:

@( myheader.h @>=

@< External typedef declarations @>
@< Prototypes of external functions @>
... etc. (see above)

and somewhere place a definition of @< Include files @> starting like

@< Include files @>=
#include "myheader.h"
#include ...
```

Note that the module @(myheader.h @> is nowhere used (in contrast with the example given earlier by Lee Wittenberg; note that CWEAVE will not complain about an unused module in such a case): there is no need to clobber the main output C(++) file with a copy of the header file that can also be accessed by #include. A external function would then be introduced by code like

```
@ Here is a useful function |do_it|.
```

```
@< Prototypes of external... @>= void do_it(int now);
@ @< Functions @>=
void do_it(int now)
{ /* ... */
}
```

This keeps the prototype right next to the function definition, and provides an easy check for the writer and the reader of the code that they do in fact match. [Incidentally, in my own version of CWEB the third line here would be written

```
@~@< Functions @>=
```

so as to prevent a page break in the printout after the prototype, but the control code `@~' is not known to Levy/Knuth CWEB; another extra of my version is that it will typeset the identifiers declared under @< Prototypes of external functions @> (or typedeffed in a header file by any other means) properly when they appear in other CWEB documents that include the header file. You can find my version in the FAQ under the name "CWEB 3.x", though the name may change in the near future.]

What are the pluses and minus of a single WEB file to generate .h and .cc? I have read the CWEB manual, and I am still not sure.

Concluding, if you are using header files for the purposes indicated above, I see mainly pluses. One minus is that the .h file gets written every time the main output file is regenerated by CTANGLE, and this may cause "make" to recompile any files that include it (since "make" cannot know whether the newly written header file is identical to the previously written one); this is similar (but worse) to the problem that "make" will rerun CTANGLE and recompile the C(++) file, even if you only changed the documentation part of your CWEB source.

From: **Lee Wittenberg**
Date: 27 Jan 1994

Tom Epperly asks: I have read the FAQ which gives a nice list of literate programming tools. However, I am wondering if someone can give me some recommendations, so I can avoid having to check them all out. I am familiar with TeX, C, and C++, and I would like to start a big project using a literate programming language. Can you recommend the top two tools that I should consider?

That's a loaded question if ever I heard one! I guess CWEB can be considered the top contender (but only because of DEK's imprimatur). Second place is up for grabs. I would vote for noweb, because that's what I use when I'm not useing CWEB, but that's just a personal choice, not a well thought-out decision.

I have looked at CWEB, and it seems nice except for its method of generating multiple output files. I need to generate an include file(.h) and a C++ file(.cc), and I am not sure about the pluses and minuses of using one or two WEB files. If I want to use a single file to generate both the .h and .cc files with CWEB, it seems that the code for the .h file must be named @(filename.h@> which isn't very mneumonic to me. What are the pluses and minus of a single WEB file to generate .h and .cc? I have read the CWEB manual, and I am still not sure.

I'm quite happy using a single web to generate both the .h and the .cc (as I've mentioned before). I generally restrict myself to a single C++ class per web with the structure:

```
@<Header files need by the implementation@>@;
@(xxx.h@>@;             // simpler than a #include
@<Members of |xxx|@>@;

@ @(xxx.h@>=
#ifndef XXX_H
#define XXX_H   // prevent multiple inclusions

class xxx {
        @<Private |xxx| members@>@;
protected:
        @<Protected |xxx| members@>@;
public:
        @<Public |xxx| members@>@;
};
```

which works quite well (I use similar techniques in noweb, and I assume that they are available in FWEB, nuweb, etc.). The only problem is that CTANGLE assumes that its output is going to a .c file. What you have to do is use the command line "ctangle xxx /dev/null xxx.cc" to get the output into a .cc file (use NUL instead of /dev/null for DOS). I believe that CWEB 3.1 has a better way of doing this, but I haven't gotten around to downloading it yet.

From: **Julian Birch**
Date: 28 Jan 1994

Tom Epperly asks: I have read the FAQ which gives a nice list of literate programming tools. However, I am wondering if someone can give me some recommendations, so I can avoid having to check them all out. I am familiar with TeX, C, and C++, and I would like to start a big project using a literate programming language. Can you recommend the top two tools that I should consider?

Go for noweb. Consider how much time it takes to learn CWEB, and then how much time it takes to learn the following:

```
<<Reference to chunk>>
<<Definition of chunk>>=
<<Extra code to add to chunk>>+=
@ Documentation
@ %def list of identifiers defined in previous chunk
```

Of course, it's a bit unix-specific, which is a pain, but it also fits the unix philosophy quite well, rather than the somewhat monolithic approach favoured elsewhere. I understand nuweb may be worth a look, but I haven't got around to it.

From: **Dave Thompson**
Date: 28 Jan 1994

Of course, it's a bit unix-specific, which is a pain, but it also fits the unix philosophy quite well, rather than the somewhat monolithic approach favoured elsewhere.

But, you know, Norman has already promised to change all that with the upcoming version 3.0! <ducking and running> <and probably dodging too!>

From: **Norman Ramsey**
Date: 30 Jan 1994

I am serious about 3.0, but I am also serious when I say I doubt it will happen this year. I have commitments to four other projects this year, only three of which I expect to get to :-) My hope for version 3 is to add no new features, but to provide all the existing features (including decomposition into filters) in a single C program. I will replace all the existing sh, awk, sed, and Icon code with C code or tcl code, and I will bundle the tcl code in with the C so there will be no auxiliary files. I hope this trick will improve portability and performance, and that is will make noweb much easier to install.

Seeking K&R-style formatting for C

From: **Wheeler Ruml**
Date: 28 Jan 1994

I've been programming in C (and objective-c) for a while, but would like to make the switch to CWEB or a similar system. I've grown accustomed to the style of C code formatting seen in K&R, and the formatting of CWEB, although beautiful, doesn't seem to follow the same rules. After much too much work spent hacking cweave, I still don't get the exact results I want. Is there any available modification to CWEB or is there any literate C system that will give me the standard formatting for C? I would prefer a system that pretty-prints and makes indices. Do I have to settle for a system that just passes the code "straight through" to a typewriter font? Thanks for any advice or pointers! PS - I'm sure it would be too much to ask that the system worked for Lisp as well...

From: **Norman Ramsey**
Date: 29 Jan 1994

Wheeler Ruml writes: After much too much work spent hacking cweave, I still don't get the exact results I want.

Welcome to prettyprinting.

Is there any literate C system that will give me the standard formatting for C?

There's a spiderweb grammar for C floating around somewhere, but it probably doesn't do what you want either. The sole virtue of spiderweb is that it makes this sort of hacking much easier.

Do I have to settle for a system that just passes the code "straight through" to a typewriter font?

Gee, last I saw, that's what's in K&R :-) :-) :-)

From: **Lee Wittenberg**
Date: 30 Jan 1994

Wheeler Ruml writes: I've been programming in C (and objective-c) for a while, but would like to make the switch to CWEB or a similar system. I've grown accustomed to the style of C code formatting seen in K&R, and the formatting of CWEB, although beautiful, doesn't seem to follow the same rules. After much too much work spent hacking cweave, I still don't get the exact results I want. Is there any available modification to CWEB or is there any literate C system that will give me the standard formatting for C? I would prefer a system that pretty-prints and makes indeces. Do I have to settle for a system that just passes the code "straight through" to a typewriter font?

If you don't want to "settle for a system that just passes the code `straigh through' to a typewriter font," your best (and perhaps only) bet at present is to use Spidery WEB to create your own C pretty-printing grammar (modifying the supplied one as necessary). When you've got it working, you can always retrofit the generated grammar code into CWEB, if you like.

PS - I'm sure it would be too much to ask that the system worked for Lisp as well...

I have a scratch Spidery grammar for Scheme. I'd be glad to send you a copy if you'd like to play with it (and, perhaps, flesh it out so that it really works).

From: **Marc van Leeuwen**
Date: 31 Jan 1994

Wheeler Ruml writes: I've been programming in C (and objective-c) for a while, but would like to make the switch to CWEB or a similar system. I've grown accustomed to the style of C code formatting seen in K&R, and the formatting of CWEB, although beautiful, doesn't seem to follow the same rules. After much too much work spent hacking cweave, I still don't get the exact results I want.

Could you be more precise about what you are missing in CWEB's formatting? Up till now I thought I thought that CWEB's way of formatting was K&R's. In fact, when I started using CWEB it actually could only typeset properly if I *wrote* K&R C (rather than ANSI) but that's fixed now. When I was developing my own version of CWEB I spent quite a bit of time making the grammar more manageable so that I could teach it some more decent formatting, in particular to break lines before opening braces rather than after them. The original style, which makes it virtually impossible to match up opening and closing braces visually, is still available under the `+u' (for ugly) option, but apparently that is still not what you want. But since the grammar includes this and other style variations based on a simple run-time rule selection scheme, it may still be instructive to study the code and maybe install your own variation; in any case the grammar rules just text strings and much more easily manipulated than in Levy/Knuth's CWEB. You can find my version at ftp.cwi.nl:pub/cweb, and it is also available from the archives (code name "CWEB 3.x").

From: **Christian Lynbech**
Date: 01 Feb 1994

For something in between true prettyprinting ala SpiderWeb and tt-only, one could consider the lgrind package. I do not remember exactly where I found it, but I can dig that up if anybody is interested. The idea is to write a short termcap like definition file that defines things like what is keywords and what delimits comments. The you can generate latex and tex versions of your programs. As indicated, the program is rather limited in the control it offers, but you do get some basic formatting (strings, keywords, comments, identifiers) with a very little effort, and the result respects indentation and such, without resorting to an all-is-in-tt-font scheme.

Make and noweb cpif

From: **Lee Wittenberg**
Date: 01 Feb 1994

*Christopher Booth writes: I have now given up using noweb, because I couldn't persuade my colleagues to join me. Part of the reason for that was that it is relatively difficult to see what is in each chunk unless you have good separation of chunks. I would have loved to use a folding editor; it might have increased my chances of persuading the group. (I *have* got Jamie Lokier's folding-mode for emacs :-)*

I'm sorry to hear that. I had a similar problem at Tipton Cole+Co. last year, but now they are happily literate programming (with noweb). All it takes is getting one other person to start using literate programming. No one believes the original pioneer, but when a second person starts gushing about the same stuff you've been saying, people start to take notice. Is there someone you can "lean on" to start literate programming, as well?

*The other reason that I found noweb difficult was because I am writing C++, generating lots of classes for one program. It is normal practice to put each class into two (or perhaps three) files, one header file, one implementation file (and one file of inline functions). With any reasonably sized project the number of files very soon mounts up, but there is no easy way to split up the noweb files; after all, the classes are all meant to work together, they are all inter-related, so I really want to keep them close together. The problem comes when using make. Each header file gets [no]tangled many times, but is never changed because I use cpif, but make seems to get confused and attempts to remake the same header file multiple times. This wouldn't be too bad if the noweb files weren't too long, but they *are*. Does anyone have any suggestions? I will summarize if I get lots of replies, so please email me.*

This is not really a problem at all. I speak from experience here, since the new DOS port of noweb (soon to be officially released) is written in C++ (for a variety of reasons, none relevant to this discussion). I put each class into a separate .nw file that generates both the .h and .cpp files. They are tangled separately, but designed to be woven together (I weave with the -n option, and use a separate doc.tex ``driver,'' but you can just put them all together on the noweave command line). Make doesn't get confused -- the rules

```
.nw.h:
        notangle -R$*.h $*.nw > $*.h
.nw.cpp:
        notangle -R$*.cpp $*.nw > $*.cpp
```

do the job quite nicely (I use filenames for my root chunks in C++ classes, and \let\nwnotused=\nwoutput. Sneaky, no?).

From: **Osman Buyukisik**
Date: 01 Feb 1994

Christopher Booth writes: I have now given up using noweb, because I couldn't persuade my colleagues to join me. Part of the reason for that was that it is relatively difficult to see what is in each chunk unless you have good separation of chunks.

Don't give up on litprog, try nuweb with auctex.

The problem comes when using make. Each header file gets [no]tangled many times, but is never changed because I use cpif, but make seems to get confused and attempts to remake the same header file multiple times.

nuweb takes care of this, files are not updated (touched) unless they are changed.

From: **Tony Coates**
Date: 02 Feb 1994

Christopher Booth writes: The problem comes when using make. Each header file gets [no]tangled many times, but is never changed because I use cpif, but make seems to get confused and attempts to remake the same header file multiple times.

Osman Buyukisik writes: nuweb takes care of this, files are not updated (touched) unless they are changed.

As with FunnelWeb too, however I don't think that his files were being updated each time, since he was using `cpif'.

From: **Norman Ramsey**
Date: 03 Feb 1994

Christopher Booth writes: I have now given up using noweb... Part of the reason for that was that it is relatively difficult to see what is in each chunk unless you have good separation of chunks.

I don't understand this remark. I suspect you are referring to some property of editing the source code. If you actually run noweave and print documents, I suspect you will find it much easier to see what is in each chunk. The "source code is unreadable" problem is well known; it plagues most literate-programming tools. I would be most interested to hear if you run across tools that do a better job in this respect.

*The other reason that I found noweb difficult was because I am writing C++, generating lots of classes for one program. It is normal practice to put each class into two (or perhaps three) files, one header file, one implementation file (and one file of inline functions). With any reasonably sized project the number of files very soon mounts up, but there is no easy way to split up the noweb files; after all, the classes are all meant to work together, they are all inter-related, so I really want to keep them close together. The problem comes when using make. Each header file gets [no]tangled many times, but is never changed because I use cpif, but make seems to get confused and attempts to remake the same header file multiple times. This wouldn't be too bad if the noweb files weren't too long, but they *are*.*

If I understand correctly, you are pleased about the way you are structuring your code, but you are encountering performance problems with make. I've encountered this problem before when building large systems. To state it succinctly, you are caught on the horns of the following dilemna:

1) If you don't use cpif, files containing class declarations (i.e. header files) get updated even when their contents don't change, resulting in unnecessary recompilations.

2) If you do use cpif, files containing class declarations are almost always out of date (from checking timestamps), resulting in lots of attempts by make to bring them up to date. (These attempts do nothing but chew up time---they have no side effects---but they occur every time you run make.)

You've chosen option (2). The only real way out of this box is to use a better build engine, like odin, but there is a hack that works well with option (2). Every so often, once you've built your target, run `make -t', which touches all the files in an appropriate order so that the time stamps show that become consistent. This is not an aesthetic solution, but it's viable because the unnecessary attempts only occur when you've touched a source file, so you can get away with running make -t only every so often, when things get ugly.

Another poster claimed that nuweb solves this problem. It doesn't. nuweb implements exactly the same semantics as cpif. It does make the problem less annoying because nuweb makes the rebuild attempt more quickly then noweb since it doesn't use shell scripts.

How do I use noweb effectively for C++?

From: **Marcus Speh**
Date: 04 Feb 1994

I was wondering whether other practitioners using literate programming for the C++ language, are still making use of the "good style" like hiding class defs in *.hh, member defs in *.cc and inline members in *.icc files -- I have noticed that if I do not post-WEB code but write in noweb, say, from scratch, I do tend to inline *everything*. Whether or not the parts are split up later is a matter put in an "assembly" chunk which I am only interested in at the very end. Ok, if someone insists looking at the tangled code, he/she will find extremely long, ugly inlined functions - effectively it is left to the compiler to reject one or the other definition. Thus, one is deprived of the PRAGMA directive (but not every compiler has got one). What do others think/do?

From: **Karel Zuiderveld**
Date: 04 Feb 1994

*Marcus Speh writes: I was wondering whether other practitioners using literate programming for the C++ language, are still making use of the "good style" like hiding class defs in *.hh, member defs in *.cc and inline members in *.icc files -- I have noticed that if I do not post-WEB code but write in noweb, say, from scratch, I do tend to inline *everything*. Whether or not the parts are split up later is a matter put in an "assembly" chunk which I am only interested in at the very end. Ok, if someone insists looking at the tangled code, he/she will find extremely long, ugly inlined functions - effectively it is left to the compiler to reject one or the other definition. Thus, one is deprived of the PRAGMA directive (but not every compiler has got one). What do others think/do?*

I am using a collection of perl scripts called perl4c++ which I wrote during the last two years. Before I started to use noweb, I used two files, namely a description file containing my class interface (but without the inline functions) and an implementation file that contains the actual C++ code. From these two files, usually with .des and .imp extensions, my perl4c++ scripts are generating all required files automagically:

```
- a .h file
- a .icc file with all inline functions
- a .tcc file with all template functions
- a .C file with the C++ code

as well as

- a .d file with all dependencies required for correct compilation of the .h file
- a script that generates a man page from a description file.
```

The basic advantage of using perl4c++ is that it is up to the Perl scripts to decide to put which information in which file, instead of deciding that while writing the code. This greatly increases code portability. I am considering writing a paper on my approach, since I feel that more people are running into the described problem while my approach seems to work for me and likely for others.

From: **Lee Wittenberg**
Date: 04 Feb 1994

*Marcus Speh writes: I was wondering whether other practitioners using literate programming for the C++ language, are still making use of the "good style" like hiding class defs in *.hh, member defs in *.cc and inline members in *.icc files -- I have noticed that if I do not post-WEB code but write in noweb, say, from scratch, I do tend to inline *everything*. Whether or not the parts are split up later is a matter put in an "assembly" chunk which I am only interested in at the very end. Ok, if someone insists looking at the tangled code, he/she will find extremely long, ugly inlined functions - effectively it is left to the compiler to reject one or the other definition. Thus, one is deprived of the PRAGMA directive (but not every compiler has got one). What do others think/do?*

An interesting series of points. The idea of *.icc files for inline members is new to me, and I think I prefer keeping the entire interface in the *.h file (I use `.h' instead of `.hh' or `.hpp', but the idea is the same -- the C++ standard seems to be moving toward eliminating extensions altogether from header file names, by the way), and I consider inline functions to be part of the interface. I also put the member defs in *.cpp files (my compiler prefers this extension -- maybe the new standard will have something to say on this issue, as well).

With respect to making everything inline, I also find that I have the tendency to want to declare all member functions inline, but I try to fight it in the interests of "cleaner design." I believe that the interface and implementation of a class should be kept conceptually, if not physically, separate (literate programming of course, allows me to keep them physically together in the web, but apart in the tangled code). I prefer to save inline functions for members that are either trivial or are aliases for other functions. For example, in the NWstring class I implemented for the DOS version of noweb, I have the following chunk (extracted to nwstring.h) declaring the += operator:

```
<<Public members>>=
NWstring& operator+=(const char *);
NWstring& operator+=(const NWstring& s)
        { return *this += s.p->s; }
```

(p->s is, of course, a char *). The const char * operator is defined in a chunk that is extracted to nwstring.cpp, as it is a "real" function rather than an alias. Well, that's one man's opinion. Anyone else?

From: **Zdenek Wagner**
Date: 10 Feb 1994

*You wrote: I was wondering whether other practitioners using literate programming for the C++ language, are still making use of the "good style" like hiding class defs in *.hh, member defs in *.cc and inline members in *.icc files -- I have noticed that if I do not post-WEB code but write in noweb, say, from scratch, I do tend to inline *everything*. Whether or not the parts are split up later is a matter put in an "assembly" chunk which I am only interested in at the very end. Ok, if someone insists looking at the tangled code, he/she will find extremely long, ugly inlined functions - effectively it is left to the compiler to reject one or the other definition. Thus, one is deprived of the PRAGMA directive (but not every compiler has got one). What do others think/do?*

I write my C++ programs in CWEB but the philosophy should be similar. The structuring of the sources depends on what I am writing. I will split my explanation to the following parts:

1. Class library. The intention is to build a library of hierarchical classes which has to be stored as a .LIB file (on MS DOS). the

appropriates parts will then be linked to the user program (or my own program) in the future. All classes will have something in common but a particular program may need only some of them. This logically leads to a requirement of having all sources in one web file and send each class (or a small group of classes) into a separate CPP file (implementation) and a separate header file (class definition and inline functions).

2. Program with reusable classes. Sometimes I write a program for which I develop some new private classes. It seems to me that the classes could be useful in the future in other programs. In order to make the classes easily reusable, I again send the class definition and the inline functions into a header file and the implementation to the CPP file. Moreover, I write it to a separate web file and either treat it absolutely independently (with independant weaving and tangling) or I include it into the main web file via @i.

3. Small program with special classes. Sometimes I need classes which will hardly be useful for anything else. If the program is sufficiently small, I do not care about spliting the tangled code to different header and CPP files. Instead I just build a single CPP file which only includes standard CPP headers.

4. Large program with special classes. If the size of the program exceeds some (ill defined) limiting value, my Borland C++ compiler may crash due to insufficient memory. Then I have to split the program into CPP and header files similarly as in 2 above. If some rules are followed, Borland C++ can use precompiled headers, i.e. the header files are compiled only for the first CPP files of the project and at the beginning of the compilation of other files the memory image of the precompiled headers is loaded at high speed. This speeds up the compilation drastically. When splitting the code into CPP and header files, the compilation time is the very thing which I have in mind.

From: **Stuart Ferguson**
Date: 11 Feb 1994

Although I do not use C++, I do write object-oriented code in C, as well as normal C code intended to be used as shared modules by other programs. In these cases, I write the interfaces, typedefs and macros to the ".h" and the implmentation to the ".c" files. The header is then included by clients of the class or module. This all seems pretty normal to me and I would expect that C++ would work more or less the same way.

I find that one of the great strengths of literate programming tools is just this ability to generate multiple output files from a single source. I only have to define an interface once in the interface spec part of the document and I can include it as a prototype in the header file and as the function declaration in the C file. Of course I only have to use a tool for this because C is such a horrible language, but that is not the only reason to want this capability. I also find it very useful to be able to include a test program for a module in the same document, and some complex or multipurpose modules may have multiple headers, like the public header and the `friend' header.

Two points

From: **Andreas Stirnemann**
Date: 04 Feb 1994

I have been trying to write literate C++ code for almost a year, using fweb. On the whole I am very pleased with the results. Being in the business of computer-aided mathematical proofs, I particularly like the possibility of adding mathematical formulae to the comments.

An example for the sort of code I develop is availabe via anonymous ftp from euclid.ex.ac.uk. It is the file necklace.w (the name refers to the mathematical construction it is supposed to deal with), in the directory pub/andreas. (The program won't compile stand-alone; I did not want to include the library files it depends on.) Now, I'd like to raise two points. Firstly: I often write code of the following form:

```
--------------------------- begin example ---------------------------------
@ Output function.
@<foo: friend functions@> =
ostream& operator<<(ostream&, foo);

@
@<function definitions@> =
ostream& operator<<(ostream& ost, foo f)
{
        /* ... */
}
--------------------------- end example -----------------------------------
```

What I find a bit cumbersome with this is that I am forced to introduce two numbered sections, although this code really deals with one single issue. Instead, I'd like to write:

```
---------------------------- begin example ----------------------------------
@ Output function.
@<foo: friend functions@> =
ostream& operator<<(ostream&, foo);

@<function definitions@> =
ostream& operator<<(ostream& ost, foo f)
{
        /* ... */
}
---------------------------- end example ----------------------------------
```

fweb does not like this at all. (It reports something like ! (TANGLE): Nested named modules. Missing '@*' or '@'?.) This may seem to be a trivial problem in connection with the above example. Let me therefore give a somewhat more elaborate one, which is taken from one of the programs I am actually using:

```
---------------------------- begin example ----------------------------------
@
@<private members@> =
static function* pphi;

@
@<global variable definitions@> =
function* pair::pphi;

@
@<initialize globals@> =
pphi = new function();

@
@<set up references@> =
function& phi = *pphi;

@
@<set up read-only references@> =
const function& phi = *pphi;

@
@<release globals@> =
delete pphi;
---------------------------- end example ----------------------------------
```

It seems excessive to me to have to introduce six numbered sections in order to deal with one single object. I don't see any syntactical reason why it should not be possible to write instead:

```
---------------------------- begin example ----------------------------------
@
@<private members@> =
static function* pphi;

@<global variable definitions@> =
function* pair::pphi;

@<initialize globals@> =
pphi = new function();

@<set up references@> =
function& phi = *pphi;

@<set up read-only references@> =
const function& phi = *pphi;

@<release globals@> =
delete pphi;
---------------------------- end example ----------------------------------
```

I'd really like to be able to emit code into several slots out of one numbered section. Thus, the operations of creating slots and emitting

code into slots would become dual. But maybe I am overlooking an issue here. Or maybe the above code is simply bad style.

Here is my second point, which is really a question. I am using the following strategy to declare classes:

```
--------------------------- begin example ---------------------------------
@
@a
@o foo.h
class foo
{
  private:
    @<foo: private members@>@;

  public:
    @<foo: public members@>@;
    @<foo: friend functions@>@;
};
---------------------------- end example ---------------------------------
```

Given this code, tangle produces a reusable header file, which is fine. But suppose I'd like to have a woven version of the header file, exhibiting, for instance, all the public member function. Does anyone have a suggestion how to achieve this?

From: **Marc van Leeuwen**
Date: 07 Feb 1994

```
--------------------------- begin example ---------------------------------
@
@<private members@> =
static function* pphi;

@
@<global variable definitions@> =
function* pair::pphi;

@
@<initialize globals@> =
pphi = new function();

@
@<set up references@> =
function& phi = *pphi;

@
@<set up read-only references@> =
const function& phi = *pphi;

@
@<release globals@> =
delete pphi;
---------------------------- end example ---------------------------------
```

It seems excessive to me to have to introduce six numbered sections in order to deal with one single object. I don't see any syntactical reason why it should not be possible to write instead:

```
--------------------------- begin example ---------------------------------
@
@<private members@> =
static function* pphi;

@<global variable definitions@> =
function* pair::pphi;

@<initialize globals@> =
pphi = new function();

@<set up references@> =
```

```
function& phi = *pphi;

@<set up read-only references@> =
const function& phi = *pphi;

@<release globals@> =
delete pphi;
---------------------------- end example ----------------------------------
```

I'd really like to be able to emit code into several slots out of one numbered section. Thus, the operations of creating slots and emitting code into slots would become dual. But maybe I am overlooking an issue here. Or maybe the above code is simply bad style.

Although I am not familiar with FWEB, I can see a reason why this is not possible in CWEB, and which probably applies to FWEB as well. All cross references are made to section numbers, and allowing a (numbered) section to contribute to arbitrarily many modules could rapidly make these references useless. Also, the module body is the basic unit for the tangling program, and so internally your example would be considered as six independent bodies, even if they were combined in the printed listing. There is no doubt that one could change the WEB programs so that your syntax would be allowed (add a code that means "start a new section but skip the commentary and don't increase the section number") but it would be an added feature, not the removal of an arbitrary restriction.

I admit thay typing `@ @<' is more work than just `@<', but it's not really a big deal, and I don't believe increasing the section number rapidly really hurts either (the upper limit for section numbers is rather generous). In fact you could interpret the `@ ' as an invitation to add a bit of commentary that applies the the (small) module body, although in such simple cases as above there is often not really much to say. There is one real practical problem though (at least in CWEB) namely that there is a more than average chance that a page break will occur somewhere in this list of small sections (since the first such section would easily fit on an already largely filled page). To that end one could add a new code to the system that is like `@ ' except that no page break will occur before the section, which is a much easier change than the one suggested above; I have done so in my CWEB (ftp.cwi.nl:pub/cweb) where it is called `@~'.

Whether your example is bad style I cannot say, although I personally seldom find any need to bind more than two sections together (but I'm using C, not C++). I tend to keep my variables as local as possible, almost to the extreme (if I have to swap two variables, the temporary variable will be defined locally to the three-statement block that does it), since I believe this greatly improves the clarity of the code; this reduces the number of occasions where your problem arises. One consequence of producing code by tangling is that it facilitates use of global things, since references to them can be localised in the source although they are scattered throughout the code; I think one should resist the temptation to consequently use globals for convenience, where (with a bit more planning) one could achieve the same effect with locals. Someone who is reading the code with the purpose of checking its correctness must have some mental picture (even if abstract) of the structure of the tangled code, and to that end globals are still omnipresent.

Here is my second point, which is really a question. I am using the following strategy to declare classes:

```
---------------------------- begin example ----------------------------------
@
@a
@o foo.h
class foo
{
  private:
    @<foo: private members@>@;

  public:
    @<foo: public members@>@;
    @<foo: friend functions@>@;
};
---------------------------- end example ----------------------------------
```

Given this code, tangle produces a reusable header file, which is fine. But suppose I'd like to have a woven version of the header file, exhibiting, for instance, all the public member function. Does anyone have a suggestion how to achieve this?

Joachim Schrod recently meentioned an alternative approach to this. In brief: write a separate source for the header file, and include it by means of `@i' in the implementation.

Numbering in CWEB

From: **Greg Fiehler**
Date: 10 Feb 1994

Being new to CWEB I have a question on how I would like to wee my weaved output to be numbered. When I reach the section:

```
@<The main program @>=
main()
{
@<Variables local to main@>
@<Process 1@>
@<process 2@>
...
@<Process n@>
}
---This would normally look like:
<The main program 5>=
main()
{
        <Variables local to main 6>
        <Process 1 7>
        ...
        <Process n N>
}
 --But I would like to see it as:
<The main program 5>=
main()
{
        <Variables local to main 5.1>
        <Process 1 5.2>
        ...
        <Process n 5.N>
}
-and numbered such as they appear later in the document.
```

This appraoch seems to make more sense to me in outlining the hierarchy of a program. I would like to also see sub units of these refered to as ex. 5.2.1. If anyone has done this or knows how it may be accomplished I would appreciate the knowledge or if anyone has objections stylistically or otherwise to this approach I would also appreciate the criticism or alternative approaches. I am just beginning to develop a consistent style for my attempts at literate programming and would like any suggestions on good approaches and things to avoid.

From: **Lee Wittenberg**
Date: 10 Feb 1994

As far as I recall the CWEB source, the numbering scheme you propose would require some pretty serious hacking (and a pretty big change file). What would you do about numbering a chunk that's used in more than one chunk? If is used in chunk 12, say, should its reference number be 5.3 or 12.x or 5.3/12.x? As to whether it's reasonable, who can say? Different people like different things. As far as I'm concerned, the purpose of the chunk numbers is to enable the reader to find the chunk definitions, and the CWEB scheme allows that pretty readily.

Aside: I should also mention that I am becoming quite fond of the new noweb chunk numbering scheme, where chunk numbers are page numbers with an alphabetic suffix: chunk 3a is the first chunk on page 3, 12c the third chunk on page 12, and so on.

From: **Marc van Leeuwen**
Date: 11 Feb 1994

Greg Fiehler writes: This approach seems to make more sense to me in outlining the hierarchy of a program. I would like to also see sub units of these referred to as ex. 5.2.1. If anyone has done this or knows how it may be accomplished I would appreciate the knowledge or if anyone has objections stylistically or otherwise to this approach I would also appreciate the criticism or alternative approaches.

As a short answer to your question: this cannot be done in CWEB, nor can it be achieved without major re-engineering. In fact the

information necessary to produce numbers as suggested is present when running CTANGLE, but the printed output is produced by CWEAVE, so you would have to make a kind of a merger between these programs (which cannot be advised to novices to CWEB:-). In fact CWEAVE does keep cross reference information as to which module is the parent of each module, which is part of the information necessary to deduce the numbers, but it does not record the sequence number among the siblings of that parent.

Apart from the difficulty to realise it, there are more fundamental objections to a scheme as you propose. First of all it is inherently ambiguous, since modules can be used more than once. The fact that modules may be defined in a sequence of different sections also presents problems: although the section numbers of all but the first defining section are not normally mentioned, they do appear in the index of module names at the end and in the "See also" line after the first occurrence; in your scheme there is no reasonable indication different from that of the first occurrence to refer to the other occurrences by.

Secondly, and most decisively, if you consider it closely your scheme is not desirable at all. Although the numbering proposed may make you look like a very organised person, it destroys any possibility of useful cross referencing. Consider for instance the section defining a module numbered 5.3.1.1.3.2.4; what cross reference would appear at the end of it? Well of course "This module is used in section 5.3.1.1.3.2" and maybe something like "See also sections 5.3.1.1.3.2.4a, 5.3.1.1.3.2.4b and 5.3.1.1.3.2.4c" (assuming our previous dilemma is solved by attaching letters to the number to discriminate continuation modules). That is hardly surprising information, and will not help anyone in locating the parent and continuations. Indeed, due to the fact that the programmer may arrange the module definitions in an almost arbitrary order (only continuations of one same module, including those of the unnamed root module, must appear in the relative order of their use in the tangled program) it will be virtually impossible to locate a module given its number, since numbers are not ordered. You will soon find the need for a simple sequential numbering of sections a la CWEB (or as noweb, see Lee Wittenberg's reply) for use in cross referencing, but then you will note that you need that same number to locate the defining occurrence of the module given a used one, but this was the very place you wanted to introduce the new numbering in the first place!

Concluding, in order to locate things, you want a simple sequential numbering, like page numbers (as in noweb) or lacking that by successive sections as in CWEB (page numbers would be difficult to get in CWEB, because they are assigned by TeX after CWEAVE has already done its job; I'm not sure how noweb can perform its trick, but it certainly seems to have advantages when previewing your woven code). I have been using manuals that had no page numbers but just (sub)section numbers like 4.2.3.1.3 (presumably because this facilitates updates by insertion or removal of pages) and it sure is a lot more difficult to locate any reference, even though these numbers are still guaranteed to be lexicographically increasing.

By the way, do you really use module names like @<Process 1@>, @<Process 2@>, or is that just an abstraction for the sake of the example? If you do use them, I suggest that you consider using more meaningful names, after all, a reader is supposed to understand the workings of the outer module at least roughly without having to look up the definitions of the modules used.

From: **Joachim Schrod**
Date: 11 Feb 1994

While I agree largely with Lee Wittenberg and Marc van Leeuwen on their analysis, I want to add another point I got from analyzing lots of WEB tools. (I'll write a tech report sometimes on it...) Definition: A TradWEB is a `traditional WEB system' that follows the model of Knuth's original WEB and does not change the structual elements available to the WEB programmer. E.g., the various CWEB flavors, SpiderWEB, MWEB, etc. My Hypothesis: Each WEB has two concurrent structures: The document structure and the refinement structure.

THE DOCUMENT STRUCTURE

It is used to present the software and the concepts behind its creation to the reader. If we don't give up with printed forms, the document creation is usually oriented along structures we know from the field of technical writing, i.e., it has (our should have ;-) sections, subsections, figures, references, index, etc. The basic documentation structure element of a TradWEBs is the `WEB section'. This is *not* the same as the `section's mentioned above. (DONG! WAKE UP! This was an _important_ statement... ;-) A WEB section is the basic unit that supports the usage of the technical writing paradigm "say it twice," once in an informal, once more in a formal way. (According to Knuth, that's _the_ essence of Literate Programming, btw.)

The importance of a WEB section is often underestimated. A collegue of mine recently noted that he finds the output of the doc option for LaTeX styles not much readable. We analyzed it, and it happened that he simply did not find the start of such a (explanation, code) unit. It was not supported by the typography of the document. This is a basic principle I demand from all good LitProg tools: It must support the clear detection of these units. They are the units of argumentation and communication with the reader. Without them, the LitProg paradigm cannot be taken advantage of, at least not to its full extent. The support is like using bold face at the start of an item list but not within running text. (IMO, one can learn alot about abstractions of presentations by reading typography textbooks.) How this is made actually, does not matter.

In traditional WEB systems the typographic support for the detection is done by vertical space and a bold number. Most probably vertical space alone would not suffice for a distinction, since it can appear within a WEB section as well (and does in real programs). Other structural elements are *not* supported by TradWEBs. The so-called "starred sections" are not structural units, but specially tagged sections that will be noted in the table of contents. I.e., this is not a new element for the document structure. But see below on a note about CWEB. One can cheet... :-)

THE REFINEMENT STRUCTURE

It is an acyclic graph where a subset of the nodes without predecessors are the start-nodes for minimal spanning trees that represent code for one file. As with every acyclic graph one has the problem how to identify nodes in it. The traditional solution is attaching a unique number to each node (that can be used, for example, in an adjacence matrix [sp?]). Traditional WEB systems use this canonical solution. Since at most one refinement can appear in a WEB section, they attach a number with each WEB section and use this number for the identification. Some numbers are unused, but this does not matter.

That one has numbered WEB sections and thereby numbered refinements in TradWEBs is not a hindrance per se to use higher-level sections (in the document sense) as well. In my opinion, a WEB language designer should *not* throw away this concept, the higher-level elements should be added! So, do we need to rewrite a TradWEB system -- e.g., CWEB -- to get this possibility, to get sections, subsections, figures, etc? No. -- Really, no. You mix document markup and refinement markup if you think this.

From the very beginning, the LaTeX style for CWEB has provided the possibility to provide subsections, subsubsections, etc., to the user. (The hierarchy may have up to seven levels.) It has even documentation to this effect. But I didn't need it, and nobody contributed a style option that does realize this possibility. From the very beginning, because I wrote this style when I had to include CWEB code in an other document (my thesis) where I had to adapt the CWEB document structure to the structural conventions used there. (Btw, this was _not_ Levy's (aka Knuth's) CWEB, this was The Real CWEB (mine, all mine). [Special sentence for Timothy Murphy, finished now.] :) :) Perhaps I'll add a respective package after I transformed the cweb style to a LaTeX2e class. Just to show that it's easy... ;-) 'Though I think the possibility to include CWEAVEd pieces in arbitrary LaTeX documents is more important.

Finally, a personal note: Please, do not read the above as a contribution to the ever on-going ``which WEB is better'' flamewar. (Please, do not start the flamewar again. :) This posting is a general reflection on the structure of WEB documents, and how to support them in tools. CWEB was just used as an example because I happened to be have done something there. And I wanted to pep up the dry hypothesis at the start with some concrete empiric statements. Actually, for those who think I'm the big CWEB fan: I *don't* even use Levy's CWEB -- I'm not doing much C programming any more, and CWEAVE munches C++ code in an inacceptable way. "When it comes to literate programming, I get irrational." --- DEK, 16 Jul 93

From: **Lee Wittenberg**
Date: 12 Feb 1994

Joachim Schrod makes the following observation: Each WEB has two concurrent structures: The document structure and the refinement structure.

This is something I had never thought about before, but now that I think about it, I have been (to some extent) organizing my webs around this principle. I think Joachim's insight is an extremely important one. I also think that the best webs are the ones in which the document structure makes the refinement structure clear to the reader. Note that this does not mean that the DS follows the RS in any way, but that the RS holds the key to the logic of the program and the DS should be laid out to explain it.

From: **Marc van Leeuwen**
Date: 16 Feb 1994

Joachim Schrod puts forward a few interesting points. Definition: A TradWEB is a `traditional WEB system' that follows the model of Knuth's original WEB and does not change the structural elements available to the WEB programmer. E.g., the various CWEB flavors, SpiderWEB, MWEB, etc. My Hypothesis: Each WEB has two concurrent structures: The document structure and the refinement structure.

THE DOCUMENT STRUCTURE

It is used to present the software and the concepts behind its creation to the reader. If we don't give up with printed forms, the document creation is usually oriented along structures we know from the field of technical writing, i.e., it has (our should have ;-) sections, subsections, figures, references, index, etc. The basic documentation structure element of a TradWEBs is the `WEB section'. This is *not* the same as the `section's mentioned above. (DONG! WAKE UP! This was an _important_ statement... ;-) A WEB section is the basic unit that supports the usage of the technical writing paradigm ``say it twice,'' once in an informal, once more in a formal way. (According to Knuth, that's _the_ essence of Literate Programming, by the way.)

I agree only partially with this. A WEB document is of course a very specific kind of document used for a special purpose, so it is not too surprising that it is not structured exactly like any run-of-the-mill technical paper. But given this special structure, the WEB sections are used in about the same way as sections are traditionally, or maybe more like subsections or even finer units (what is the usual name for document units that are typically about half a page or less?). The numbering structure is a bit unusual, but we are just coming to that:

The importance of a WEB section is often underestimated. A collegue of mine recently noted that he finds the output of the doc option for LaTeX styles not much readable. We analyzed it, and it happened that he simply did not find the start of such a (explanation, code) unit. It was not supported by the typography of the document. This is a basic principle I demand from all good LitProg tools: It must support the clear detection of these units. They are the units of argumentation and communication with the reader. Without them, the

LitProg paradigm cannot be taken advantage of, at least not to its full extent. The support is like using bold face at the start of an item list but not within running text. (In my opinion, one can learn a lot about abstractions of presentations by reading typography textbooks.) How this is made actually, does not matter.

In traditional WEB systems the typographic support for the detection is done by vertical space and a bold number. Most probably vertical space alone would not suffice for a distinction, since it can appear within a WEB section as well (and does in real programs).

Good points, I agree fully here.

*Other structural elements are *not* supported by TradWEBs. The so-called "starred sections" are not structural units, but specially tagged sections that will be noted in the table of contents. I.e., this is not a new element for the document structure.*

Why do you deny "starred sections" the qualification of structural units? In fact of course it is not so much the starred section by itself, but rather the whole range of sections until the next starred section that forms a unit. In my opinion these are much like chapters of the literate program: they start on a new page, get a bold title, which also goes to the running heads during the whole range of sections, and to the table of contents, all very traditional it seems to me (note that the _number_ of the starred section gets none of the special treatments its title gets, which agrees with the view that the first section itself is in no way special). Indeed, if you look in "TeX, The Program" or "METAFONT, The Program", you will find that these ranges of sections are called `parts', and get some even more special treatment, such as a separate level of numbering.

THE REFINEMENT STRUCTURE

It is an acyclic graph where a subset of the nodes without predecessors are the start-nodes for minimal spanning trees that represent code for one file.

Sounds too learned for me, but I can see what is intended. In any case you mean _directed_ acyclic graph.

As with every acyclic graph one has the problem how to identify nodes in it. The traditional solution is attaching a unique number to each node (that can be used, for example, in an adjacency matrix). Traditional WEB systems use this canonical solution. Since at most one refinement can appear in a WEB section, they attach a number with each WEB section and use this number for the identification. Some numbers are unused, but this does not matter.

*That one has numbered WEB sections and thereby numbered refinements in TradWEBs is not a hindrance per se to use higher-level sections (in the document sense) as well. In my opinion, a WEB language designer should *not* throw away this concept, the higher-level elements should be added! So, do we need to rewrite a TradWEB system -- e.g., CWEB -- to get this possibility, to get sections, subsections, figures, etc? No. -- Really, no. You mix document markup and refinement markup if you think this.*

Still I believe the higher-level elements you mention cannot be considered to all be of the same kind. For things like figures and references, that would comfortably fit into individual WEB sections, there is no real problem: you can just include them in the commentary part (in fact most of my literate programs end with a section headed by "@* References and Index." followed by a list of references just like my math papers, albeit a bit shorter; it works just fine). To add structural units that _contain_ several WEB sections is quite a bit more difficult.

From the very beginning, the LaTeX style for CWEB has provided the possibility to provide subsections, subsubsections, etc., to the user. (The hierarchy may have up to seven levels.) It has even documentation to this effect. But I didn't need it, and nobody contributed a style option that does realize this possibility.

I don't suppose you mean subsections and subsubsections as subdivisions of WEB sections, although that would explain why nobody needed them. Otherwise, I would like some explanation of the basic mechanism, which would interest me even though I don't use the LaTeX style. As far as I can see, WEB is rather unfriendly when it comes to adding textual stuff that goes outside the WEB sections, e.g., chapter titles (at a level coarser than given by the starred sections), except at the very beginning where one can use the limbo section: the problem is that there is no place for TeX code between the code part of a previous section and the section number of the next one. Even at the beginning of the document I have found myself going through strange bends in order to get a large centered title above the first section (and on the same page); eventually I started with an unstarred section to prevent a page break, and fiddled a bit to get proper running heads before the first real starred section came along. To get similar document structure once the first section is given, I can see only one possibility: to redefine the macros `\M' and (particularly) `\N' used by CWEB to start a new (starred) section, to the extent that, possibly depending on an argument, they will output some things before they give the section number. As far as I can see the newest version of CWEB already goes a bit in this direction with its support for sections starting like `@**', although one still has to do some advanced macro writing to get the kinds of effects I indicated. I would be interested if anybody has other suggestions to handle these matters conveniently.

From: **Christian Lynbech**
Date: 28 Feb 1994

Marc van Leeuwen writes: A WEB document is of course a very specific kind of document used for a special purpose, so it is not too surprising that it is not structured exactly like any run-of-the-mill technical paper.

I actually experienced some difficulties with this. First let me explain that I'm still in the midst of my first major project using the literate programming paradigm, and that I used first nuweb and now noweb, so all I'm about to say may just be a beginners inexperienced fumbling about with the wrong tools.

Anyway, my initial approach, using nuweb, was to write my program just as I would write a report. In fact, what I am writing is supposed to *be* the report. And for quite some time I was surprised that literate programming wasn't more convincing. I was slowly reaching the conclusion that Joachim put into words, My own formulation would then be, extrapolating Joachims observations: Writing a report is *different* from writing a program.

Note that with report I do not necessarily mean something of publication standard, but just something that one could consider giving ones advisor as conclusion of a project. I would not hesitate to embed code in such a thing, though my advisor certainly would not be especially interested in the code parts. So readability is of a higher concern than for something like the Stanford GraphBase webs.

If one reads Knuth's original article on WEB, he writes that it allows him to write almost in a "stream of consciousness". And I longed for the freedom that WEB's lowlevel @ sectional unit command was giving. But while stream-of-consciousness certainly helps the programmer, it is not a good strategy for report writing, in my opinion. You need to carefully plan the order in which you present the material to your reader, and things that may be developed together, may best be put in entirely different parts of the report.

On top of this, neither nuweb nor noweb enhance the WEB section structure with typographical support, which I have come to regard as a major shortcoming. They do typeset code sections well, but the code tends to float around as small islands in a sea of surrounding text, making it difficult to isolate the corresponding documentation part for a WEB section.

To some it may be obvious, but the fact that *programs* cannot be easily incorporated in *reports*, was new to me. What do others think?

From: **Andrew Mauer**
Date: 07 Mar 1994

Christian Lynbech writes: On top of this, neither nuweb nor noweb enhance the WEB section structure with typographical support, which I have come to regard as a major shortcoming. They do typeset code sections well, but the code tends to float around as small islands in a sea of surrounding text, making it difficult to isolate the corresponding documentation part for a WEB section.

I'm not exactly sure what you are saying. Both nuweb and noweb provide the standard LaTeX formatting commands, which I have found to be mostly adequete for document formatting. At times I find myself wishing for a another level of depth, but it is not usually missed. My standard progression is \section, \subsection, \subsubsection, \subsubsection*. Things which are so unimportant that they do not fit in their own subsubsection really do not seem to be large enough details to bother the reader with in the table of contents. In practice, I like the way this turns out. If, on the other hand, you are complaining that the chunks are not typographically integrated with the text, I'm not sure whether I agree, nor am I sure that separation is not a good thing.

From: **Craig Groeschel**
Date: 09 Mar 1994

Christian Lynbech writes: But while stream-of-consciousness certainly helps the programmer, it is not a good strategy for report writing, in my opinion. You need to carefully plan the order in which you present the material to your reader, and things that may be developed together, may best be put in entirely different parts of the report.

Yes. Once, before I let my colleagues inspect a short module I had written, I found myself going back through it, rearranging chunks, saying, "It would make more sense to explain this here." I wondered if that indicated a flaw in my thinking, that if I thought _logically_, and really kept my audience in mind, I would not have to rearrange.

From: **bbeeton**
Date: 10 Mar 1994

Craig Groschel wonders if rearranging chunks of a program module "indicated a flaw in my thinking, that if I thought _logically_, and really kept my audience in mind, I would not have to rearrange." If it's any comfort, I heard Don Knuth admit that when he was writing

the texbook, he realized he couldn't find a way to explain something logically, so instead he went back and rewrote the program code so that a clear explanation was possible.

Emacs mode for noweb

From: **Karel Zuiderveld**
Date: 11 Feb 1994

I'm doing most of my programming in noweb. When writing the actual code chunks, I want to use the c++-mode, while I'm using the latex mode of auctex for writing the docs. So I actually want to use two modes in Emacs kind of simulteanously. So if I write << text >>= <CR> the editor switches to c++ mode and when typing @ at the start of the line emacs switches to latex mode. You get the idea. Has someone out there written (or can somebody write) a hook that allows for automatic setting of modes?

From: **David Plummer**
Date: 11 Feb 1994

[[This question was originally on comp.programming.literate. I'm crossposting this to gnu.emacs.help, since I suspect that code to do what is suggested might already exist. If anyone there knows of Emacs code to combine two modes, I (and I assume many on comp.programming.literate) would be interested to hear about it. Since this is basically a gnu.emacs question I have directed followups to gnu.emacs.help. I will summarize subsequent discussion there on comp.programming.literate if there is any. Apologies to readers of gnu.emacs.help if this is not the appropriate forum for this posting.]]

Karel Zuiderveld writes: I'm doing most of my programming in noweb. When writing the actual code chunks, I want to use the c++-mode, while I'm using the latex mode of auctex for writing the docs. So I actually want to use two modes in Emacs kind of simulteanously. So if I write << text >>= <CR> the editor switches to c++ mode and when typing @ at the start of the line emacs switches to latex mode. you get the idea.

The question posed is a special case of the question; how can I choose the major mode on the basis of the contents of the file, and my position relative to that contents? Here is how I would proceed to build a mode which combines two different modes. I will implement this eventually, unless I find that it has already been done, so if anyone wants to shoot down, or suggest improvements on this strategy, I'd appreciate receiving comments.

Basically, you need to build a new keymap for the combined mode which has two types of entries: unconditional and conditional. - Unconditional entries occur whenever the two keymaps agree on the function to be called, and the entry is just a call to that function. - Conditional entries point to a function which looks around the point to determine which context we are in, and then calls the function from the appropriate keymap.

That is, suppose that in C++ mode, key Ctrl-A calls function foo and in LaTeX mode it calls function bar. Then the combined mode would call foo-or-bar, defined to be: (if (c++-context-p) (foo) (bar)) The trick would then be in the definition of c++-context-p. I think that this would work pretty well when the keymaps mostly agree or at least when the places where the keymaps disagree are used relatively infrequently. I suspect that this is the case for many literate programming contexts.

PS: I don't think that it would be appropriate to design the combined mode in the way that the original poster described. Since, this scheme assumes that the file is edited from beginning to end, and that one never moves back to revise text which has already been entered. If the way in which Emacs decided which mode to be in was the immediately preceding keystrokes, then movement commands would make it possible for the mode and context to disagree.

From: **Julian Birch**
Date: 14 Feb 1994

Karel Zuiderveld writes: So I actually want to use two modes in Emacs kind of simulteanously. So if I write << text >>= <CR> the editor switches to c++ mode and when typing @ at the start of the line emacs switches to latex mode. you get the idea. Has someone out there written (or can somebody write) a hook that allows for automatic setting of modes?

Please don't flame me if I'm incorrect, but... I don't think it's that simple, sadly. I find that switching between modes for noweb doesn't work as well as expected - c-mode reads <<chunk name>>= and thinks the current line is a continuation of an assignment (>>= being an assignment operator) and hence gets the indentation horribly confused. Hope this helps at least explain why no-one's done it yet.

From: **Tony Coates**

Date: 15 Feb 1994

I have been thinking about the same thing for FunnelWeb. My current solution, which I haven't written yet, is to implement a minor mode for creating a `mode-stack', something like the directory stack often used in UNIX shells. It would be up to the user to push and pop the correct modes onto and off the stack, but an empty push would swap the two most recent on the stack. What do you think of this idea? Not that I've tried it out yet, but it seemed to me to be an achievable suggestion.

From: **Joachim Schrod**
Date: 15 Feb 1994

Tony Coates writes: I have been thinking about the same thing for FunnelWeb. My current solution, which I haven't written yet, is to implement a minor mode for creating a `mode-stack', something like the directory stack often used in UNIX shells. It would be up to the user to push and pop the correct modes onto and off the stack, but an empty push would swap the two most recent on the stack.

Why shall it be done by hand? To repeat myself, from a posting I sent only to g.e.h: In Lucid Emacs one can use the concept of extents to attach attributes to a text area. (GNU Emacs 19 has a similar concept, 'though I don't know its name.) If the position is changed one checks if the extent has changed and then one changes the mode. The actual problem is that AUC-TeX does initialization and finalization actions. They have to be done properly as well, this must be organized. Otherwise it's rather easy, and fast in addition. Oh yes, and all the other tools: etags, font-lock, func-menu, etc., must be adapted as well... They all depend on the one-mode-per-buffer-or-file paradigm...

This approach is fast. A file content gets its extents attached on read in, and on-the-fly while one types. In noweb that's a bit more difficult than in other webs since the refinement delimiters are matched by heuristics and have meaning in other modes as well. (Read: I've never understood why <<...>> is so much better than @<...@>, especially as it's well known that the former construct appears quite often in many languages [not only in C].)

As outlined above, the real difficulties are somewhere else. After all, we want to use the rest of Emacs as we're used to, don't we? PS: If anybody makes yet another web mode: Please, don't bind \C-C LETTER like the web-mode does. Or like cweb.el, that even binds keys globally.

From: **Kayvan Sylvan**
Date: 15 Feb 1994

Dominique de Waleffe's nuweb-mode does a good job with this for nuweb. It starts out in LaTeX mode. Within a scrap, you can hit C-c C-z to edit it. This brings up another buffer, where the scrap is copied and put into the appropriate source mode (the source mode is user-definable, since nuweb is language independant). Once you're done with the scrap, you hit C-c C-z again to install it. It has other conveniences as well. Perhaps someone could use the same approach for noweb or FunnelWeb.

From: **David Kastrup**
Date: 16 Feb 1994

Joachim Schrod writes: This approach is fast. A file content gets its extents attached on read in, and on-the-fly while one types. In noweb that's a bit more difficult than in other webs since the refinement delimiters are matched by heuristics and have meaning in other modes as well. (Read: I've never understood why <<...>> is so much better than @<...@>, especially as it's well known that the former construct appears quite often in many languages [not only in C].)

The <<>> solution is just a bit more appealing to the eye, if you are reading the source by human. I do agree this looks a bit nicer, but might cause more puzzling than necessary when reading texts which use, for example, << and >> operators. Personally, I do not find the choice that bad. Perhaps this is because I am desensibilized by C++. I do remember, however, that reading the first time "template <class T> { ..." I had the urge to flame somebody, who obviously never had taken the pains to write a parser without using an automated tool. This still reverbs in compiler writers thinking about how they can handle "xxx <23> z;" and there are lots worse examples.

The noweb solution, obviously, has something to do with being matched by awk, where you can somewhat reasonably do this on a line by line base the way noweb does. Whether it was a good design decision, well... Up to know it has not disturbed me, but it might lead to problems with computer generated code/computer handled syntax operations as with Emacs.

From: **Christian Lynbech**
Date: 16 Feb 1994

Karel Zuiderveld writes: I'm doing most of my programming in noweb. So if I write << text >>= <CR> the editor switches to c++ mode and when typing @ at the start of the line emacs switches to latex mode. Has someone out there written (or can somebody write) a

hook that allows for automatic setting of modes?

Thus spake Norman Ramsey: Kostas OIkonomou has done something like this for lucid emacs. If I ever get a chance to install lucid and test it, I'll include it in the noweb distribution. Meanwhile you might want to check with Kostas; I think his email address is ko@surya.ho.att.com

Someone else did a web mode (can't remember, but I think it was for nuweb). I never got round to testing it, but the idea was that you would edit each chunk in a separate buffer (just like mail and RCS logs), using some scheme like the usual `-*- c -*-' mode setting feature, for setting the major mode of the buffer, maintaining the original major mode for the web buffer. It was posted to the net, so somebody probably knows where it can be found.

My own approach has been to overload the indentation function. I have made a small script that allows me to use environment syntax:

```
\begin{scrap}{This defines foobar}
int
foobar()
{ <stuff> }
\end{scrap}
```

Then I can make a small function that checks whether it is inside such an environment or not, using either one of the appropriate indentation functions. This scheme can be more or less sophisticated, and uses that you must have an indentation function that can work locally. This seem to be the case of the ordinary ones such as lisp and c, but you may have to rip the indentation function from some major modes and make a minor mode out of it. But it can be done, and I havbe found ot works reasonably. I can post some of the interesting functions, if anybody cares to see.

From: **Jacob Nielsen**
Date: 16 Feb 1994

I`ve followed the discussion on an emacs mode for doing literate programming in noweb. I use nuweb, but the main point is that there is a emacs mode for nuweb, that allows you to edit the code parts in a seperate buffer using c/c++/lisp or whatever mode you have. The code is not that hard to follow (being fairly iliterate on elisp myself, that`s a compliment to the author :) This approch works very well (IMHO) The only requirement is that you are using emacs version 19.xx. It`s included with nuweb (since I can`t remember the ftp-location at the moment -- anyone ? -- anyone interested can email me for a copy) BTW, the nuweb-mode works together with the AUC-TeX-mode but I allso have a changed version that doesn`t rely on AUC-TeX, so no need to dispair if AUC-TeX is not installed at your site.

From: **Norman Ramsey**
Date: 17 Feb 1994

Joachim Schrod writes: [noweb is harder to recognize because it uses heuristics]

I don't understand what this statement means. There's a perfectly precise algorithm for identifying noweb markup...

[and because the delimiters can be used meaningfully in other contexts]

An interesting point. I never considered the advantages of restricting delimiters to be permitted only in one context. Obviously it would much simplify the construction of an emacs mode. It's worth mentioning that noweb comes with a separable parser that reads all the markup, so if you want to write a batch tool that does something amusing, it's easy. Sadly, this helps nobody edit.

I've never understood why <<...>> is so much better than @<...@>, especially as it's well known that the former construct appears quite often in many languages [not only in C].)

For what it's worth, this was my rationale for abandoning the @<...@> standard:

1) symmetry: <<...>> is a palindrome, appropriate since << and >> are brackets. I considered but rejected @<...>@.

2) ergonomics: I have difficulty typing @<...@>; I find <<...>> much easier

3) typography: <<...>> seems to catch the eye better, perhaps because it leaves lots of white space. This is important when editing the source. I chose [[...]] over |...| for similar reasons; it's important to be able to spot these things when editing. (Plus it's easier to detect a missing delimiter when opening and closing delimiters don't match.)

David Kastrup writes: The noweb solution, obviously, has something to do with being matched by awk, where you can somewhat

reasonably do this on a line by line base the way noweb does.

Interesting exegesis, but actually it had nothing to do with awk. Human beings are very good at perceiving line breaks and indentation as significant, and I wanted to take advantage of that. I ended up by not using indentation to terminate code chunks, however, because I decided the rules for determining the refinement would be too complex. (You will notice, however, that noweave uses indentation to mark the scope of code chunks.)

From: **Norman Ramsey**
Date: 18 Feb 1994

Jacob Nielsen writes: there is a emacs mode for nuweb, that allows you to edit the code parts in a seperate buffer using c/c++/lisp or whatever mode you have.

This scheme strikes me as a loss. I want the documentation parts right there in my face (in the same buffer as the code) so I'm likely to keep them consistent. Documentation together with code, even when editing, is one of the strengths of the literate-programming paradigm.

From: **Tony Coates**
Date: 18 Feb 1994

Jacob Nielsen writes: there is a emacs mode for nuweb, that allows you to edit the code parts in a seperate buffer using c/c++/lisp or whatever mode you have.

Norman Ramsey writes: This scheme strikes me as a loss. I want the documentation parts right there in my face (in the same buffer as the code) so I'm likely to keep them consistent. Documentation together with code, even when editing, is one of the strengths of the literate-programming paradigm.

Myself, I have just started working on an emacs mode for FunnelWeb. At this (rather early) stage, if you are just linearly typing, as you type @{ to start a code fragment, it prompts you for a major mode, allowing normal name completion from all the possible modes, and using the last chosen mode as a default. When you type @} to finish a code fragment, it returns the mode to the chosen (AUC/La)TeX mode. I still have a lot of work to do detecting movement in and out of code sections, but I'll try to get to that soon. Anyone have any ideas/suggestion?

From: **Dominique de Waleffe**
Date: 18 Feb 1994

Jacob Nielsen writes: there is a emacs mode for nuweb, that allows you to edit the code parts in a seperate buffer using c/c++/lisp or whatever mode you have.

Norman Ramsey writes: This scheme strikes me as a loss. I want the documentation parts right there in my face (in the same buffer as the code) so I'm likely to keep them consistent. Documentation together with code, even when editing, is one of the strengths of the literate-programming paradigm.

To be more precise than Jacob, what the mode for nuweb does it that it takes a scrap contents into a separate buffer just for editing that scrap then puts it back in place in the document. Which you still see. Thus, Norman, I don't think that is such a loss. That allows to use all the specific facilities of C/Lisp/Makefile etc modes for editing the portions of code. But globally, one is editing the nuweb file in a LateX-mode buffer with the whole thing available. One of these days when my project leaves me a few hours free, I'll put some more work into this mode. [I know I've been saying this for months...]

From: **Przemek Klosowski**
Date: 18 Feb 1994

Jacob Nielsen writes: there is a emacs mode for nuweb, that allows you to edit the code parts in a seperate buffer using c/c++/lisp or whatever mode you have.

Norman Ramsey writes: This scheme strikes me as a loss. I want the documentation parts right there in my face (in the same buffer as the code) so I'm likely to keep them consistent. Documentation together with code, even when editing, is one of the strengths of the literate-programming paradigm.

In practice it is not such problem because the elisp code splits the window and displays the code in the bottom part. If your code is over the regulatory 12 lines, and the documentation part is similarly long, it actually is a win, because you get to scroll around the code without scrolling the text out of sight.

From: **Lee Wittenberg**
Date: 18 Feb 1994

Joachim Schrod writes: I've never understood why <<...>> is so much better than @<...@>, especially as it's well known that the former construct appears quite often in many languages [not only in C].)

Norman Ramsey writes (in response to Joachim Schrod's comments):

For what it's worth, this was my rationale for abandoning the @<...@> standard:

1) symmetry: <<...>> is a palindrome, appropriate since << and >> are brackets. I considered but rejected @<...>@.

2) ergonomics: I have difficulty typing @<...@>; I find <<...>> much easier

3) typography: <<...>> seems to catch the eye better, perhaps because it leaves lots of white space. This is important when editing the source. I chose [[...]] over |...| for similar reasons; it's important to be able to spot these things when editing. (Plus it's easier to detect a missing delimiter when opening and closing delimiters don't match.)

As a sidelight here, I've been using literate programming notation in my introductory programming classes to help demonstrate step-wise refinement. I tell the students that the English text inside the <<>> brackets are "placeholders" for bits of code that we'll write later. When I actually write the code chunks, I use the <<>>= notation.

I chose to use <<>> instead of @<@> because it seemed easier to explain to a bunch of novices. I could just hear the crazy questions about the at signs and what they were used for. On the other hand, I've discovered that when I write <<...>> on the board, no one bats an eye -- they're obviously brackets (whereas @< and @> are only brackets to the initiated).

This is the second semester I've been trying this experiment, and it seems to be succeeding. Students have no trouble "tangling" the code themselves, and their notes now contain information about the *process* of stepwise refinement, rather than just a completed program or a bunch of code fragments with arrows pointing every which way (to show where in the program the code belongs). On the other hand, they say that "all educational experiments are doomed to succeed."

From: **Christopher Forkin**
Date: 23 May 1994

I hope this isn't a FAQ. I use CWEB and SchemeWEB lately and haven't been able to come up with a useable solution to use them with emacs. I'm used to running emacs in its different (lisp-mode, auctex-mode, c-mode, ..) and would like to have something like a cweb-mode and a schemeweb-mode, which would be like c-mode in the c-context and auctex-mode (or similar) in the tex-context and so on. Does anybody know of such a beast?

How do you write C++ classes in CWEB?

From: **Jerome Chan**
Date: 11 Feb 1994

I'm in need of an example on how to use CWEB to write C++ classes. The main problem I have is with seperating the file into the *.h and the *.c files. I can't seem to find an elegent way of doing it. Can any CWEB guru give a hand? Currently, this is the way I'm doing it.

```
@q Introduction/Overview @>
@* Introduction. Item is a C++ container class.
The class is defined in the following files:

@c
@(Item.h@>; /* Header File */
@(Item.cpp@>; /* Class Defination */
@(ItemTest.cpp@>; /* Test Code */

@q Headers @>
```

```
@* Headers. The headers are defined in |Item.h|.

@(Item.h@>=
@<Pragmas and Defines@>;
@<Include Files@>;
@<Class Declarations@>;

@ Something relevent.
@<Pragmas and Defines@>=
#pragma once
#define foot smelly

@ Something interesting.
@<Include Files@>=
#include <iostream.h>

@ Something fresh.
@<Class Declarations@>=
class Item {
public:@/
@<public stuff 1@>;
private:@/
@<private stuff 1@>
};

.
.
.

@* Class Definations.

@<Item.cpp@>=
@<public stuff 2@>;
@<private stuff 2@>;
.
.
.
```

The main problem is the @<public stuff x@> declarations in the .h file and the class defination in the .c file. Is there a way I can use one @<public stuff@> in the .h and the .c file? Any helpful pointers or hints? I really like using CWEB.

From: **Lee Wittenberg**
Date: 12 Feb 1994

*Jerome Chan asks: I'm in need of an example on how to use CWEB to write C++ classess. The main problem I have is with seperating the file into the *.h and the *.c files. I can't seem to find an elegent way of doing it. Can any CWEB guru give a hand?*

I've mentioned this before, but the structure I like to use in my CWEB programs is (explanations are contrived, and should not be considered in any way a good example):

```
@*A Sample Class.
Here we define the |xxx| class. The interface will go in the
\.{xxx.h} file.
@s xxx int      @q I don't always trust the parser @>
@(xxx.h@>=
#ifndef XXX_H
#define XXX_H   // prevent multiple inclusions
@#
@<Definitions, if any, needed by the interface (may include header
files@>@;
@#
class xxx {
    @<Private |xxx| members@>@;
protected: @/
    @<Protected |xxx| members@>@;
public:    @/
```

```
    @<Public |xxx| members@>@;
};
@#
#endif

@ The members themselves are defined in the ``unnamed chunk.''  I let
\.{CTANGLE} generate \.{xxx.c} (or \.{xxx.cc} or \.{xxx.cpp},
depending on the compiler's preference).
@c
@<Header files@>@;
@(xxx.h@>@;     // included by \.{CTANGLE} rather than \&{#include}
@<|xxx| members and friends@>@;

@ To show you how I define a member, I'll make a simple constructor,
that takes a string parameter. First, I declare it (for the
interface):
@<Public...@>=
xxx(const char *s = NULL);

@ Then I define it for real:
@<|xxx| members...@>=
xxx::xxx(const char *s)
{
    @<Initialization needed for an |xxx|@>;
}

@ Let's have a data field that we can do something with.
@<Private...@>=
char *dummy;

@ And we'll initialize it in the constructor:
@<Initialization...@>=
dummy = s;

@*Epilogue.
This technique works quite well for me, and I find it reasonably
``elegant,'' as well as flexible enough to accomodate a wide variety
of classes.
```

Why do these questions keep popping up here?

From: **Mary Bos**
Date: 03 Mar 1994

Wei Lu writes: I am having hard time in figuring out the way to call TSR from a windows program via interrupts. I will appreciate any comments and references to my problem.

Tommy McGuire writes: Why do these questions keep popping up here? Is there some meaning to "literate programming" that I'm not aware of? There have been too many of them for this to be coincidence.

In Mr. McGuire's questioning the extraneous questions, I agree there are entirely too many of these types of postings. Perhaps we can use this an educational opportunity, e.g. have a standard message the literate programming group sends to the requestor and possibly to the computer science and computer services of the institution (many of these requests come from schools) about what literate programming is (and what we aren't - a source of program fragments, techniques, or hacks (in the sense of tricks)). The return message could be simple, send the path to the faq.

I'm a subscriber and not a news reader, but is there some description included in what this newgroup is about? Also there should be some copyright, copyleft, or whatever protection notice of programming examples posted in this news group. nec.com implies a commercial concern to me and so should be careful about using code fragments or programs from outside sources. I hope any programming answers we give to these types of programming requests are in a WEB for easy understanding and reading.

My dictionary gives several definitions of literate:

a: able to read and write

b: versed in literature or creative writing

c: well-read

d: having knowledge or competence <computer-literate>

So, maybe these people are reading the dictionary and figure we have flights of creativity about their problem?

From: **Mary Bos**
Date: 17 Mar 1994

To learn the best about programming style in general (and organization) - read the woven CWEB code available on the various servers. (See the FAQs in the newreader for comp.programming.literate) You learn to appreciate what you can do and see how others write the same code. If you aren't fond of C, try the original WEB written by Knuth - as a maintenance programmer I find this is the ultimate in maintainable code (and I don't care for pascal). Most of programming is not the actual composition of the syntax but the meaning behind the code, the design, and suitability of the code to the task. Once you grasp this, learning the syntax is slugging it out with the compiler, having a library reference, and practice.

Note: This is not only a newsgroup but some of us are listsubscribers.

From: **Mary Bos**
Date: 02 Jun 1994

Mark Sundermeyer writes: Clarifier to Help! Basically, the occupational problems I might run into if I were to become a programmer.

Occupational problems I have seen and had are:

1> Not keeping up with the technology - someday you may have to find a new job within or outside of the current company.

2> Not expanding your skill set. There are quite a few COBOL programmers wandering the streets right now. Twenty years ago this was the language (I am an engineering programmer, so my first language was FORTRAN, closely followed by assembly languages) and it looked to keep those practicioners employed for 50 years.

3> In some groups, there is very low socialization. If you are more people oriented - you will feel like you have been exiled to a silent cloister. On the other hand, if you aren't people oriented and choose to work as a support person, you will find yourself extremely disliked by the client population even if you extrememly competent at solving the technical problems.

4> Management and programmers have some cultural clashes. If you are a good programmer and want to step up into the management, there is quite a shock about managing the communications challenged programmers. Many programmers I have worked with are brilliant but aren't necessarily team players with an authority figure but work better in a team where everyone has their expertise (peer teams) and have an organization support person.

5> Many programmers only work 3 years as a programmer with an eye on becoming a manager. It's just something to do to step up the ladder. The better programmers take two divergent paths, one group moves on into the problem domain to research and create solutions beyond the computer and the other group (such as myself) specialize and become programmers by advocation.

6) The first couple of jobs/tasks you will have as a programmer are in maintenance. Most find this to be pure drudgery and want the more sexy projects & new development. My speciality is maintenance programming (I enjoy fixing broke code, stamping out problems, and adding new features to make the tool better fit the job. My family tradition is full of toolmakers, mechanics, and build restoration crafts). Much of the code you look at was alright when it was new, but is in abysmal shape now - extremely "work hardened" and very fragile. If you are lucky the comments and documentation you can find match what is going on and explain the philosopy of the application, program, or code chunk. If you are unlucky, you have a lot of comment and documentations that are meaningless (if luckier) or don't reflect reality (if especially unlucky) and there is no one around to explain why the strange machinations are present. Legacy systems can be soooo fun. The nastier the system, the more likely newer programmers will be assigned to it.

7) My advice from one who has been there - become a domain specialist first and a very competent programmer as a close second. Programming as it is today is rapidly changing. In 20 years there may be little resemblence to the programming today. The domain will probably exist. Learn more than how to cut code for "one-shot" projects done in school. Learn such techniques as structured methods (design and coding), and object-oriented techniques (analysis, design, and coding). Take a friend's "one shot" program for a class and add an enhancement with the rule, you can't ask your friend what the program does, why it does what it does, and any other helpful hints. If you can successfully do a major enhancement without breaking the code in less than time than the friend took to write it (the friend's time does not count debugging time and testing time while your time total includes debugging and testing) and construct a set of tests to run on the program before and after to insure your enhancements are working correctly and the original functionality is still working - then I'd say you'd have a good idea of your first 6 months to 2 years of programming. If you want to simulate reality better, have the friend give a short nondescriptive paragraph of the enhancement the friend wants added to the program and let the friend decide after you finished with changes and testing if you matched the friend's perception of the needed enhancement.

8) The age old question - what's my work environment like - I've worked from having executive office for myself to a stuffy room 10' by 4' room with 7 people (that really didn't like each other) and 1 telephone. Other friends of mine have worked in a large room with 5 desks shoved side by side in a row with 10 to 20 rows in a room. Don't have gastric problems if you aren`t blessed with an aisle seat. Currently I work in an open office, I have a 10' x 10' cubicle with no top and no door, it can be a bit noisy with other conversations nearby but it's fine.

Again, I'd recommend "The Psychology of Programmers" by G. Weinberg for some of what you may meet. "Old programmers never die they just get stuck in an infinite no op loop"

News from Phoenix (ACM conference)

From: **Lee Wittenberg**
Date: 09 Mar 1994

Greeting from Phoenix! Literate programming seems to be picking up steam, slowly but surely. Addiston-Wesley has The Stanford Graphbase prominently displayed and a hugh stack of The CWEB System of Structured Documentation copies that they are giving away right and left (I'd estimate that only about a third of the original pile remains). Other publishers seem very interested in LP, even though they don't have any books, as yet (this is a major improvement over last year, when all I got were blank looks).

I ran into an editor from Gordon and Breach Science Publishers, a firm that specializes in technical journals. He was quite excited about the idea of a refereed journal for literate programs (an idea we discussed on the net sometime within the last year). More on this if anything actually comes of it. On Thursday, Stephen Shum and Curtis Cook are presenting a paper entitled ``Using Literate Programming to Teach Good Programming Practices.'' More on this after their talk -- there's a long line waiting for these terminals.

From: **Lee Wittenberg**
Date: 20 Mar 1994

I've been back from Phoenix for over a week now, and I'd better get this report written now, or I'll never do it :-).

```
\begin{list}

\item[The CWEB book]
This book went like hotcakes. Out of 125-150 copies available on
Tuesday (the A-W people didn't know exactly how many), less than a
dozen were left by Thursday afternoon, and they were all gone by 11AM
on Friday. I couldn't have been responsible for more than about a
dozen of these, so interest in LP seems to be growing (of course, it
could have been the ``Free Free Free'' sign on top of the stack that
people couldn't resist).

While walking past the Borland booth, I overheard a couple of people
ask about C/C++ prettyprinters, and I directed them to the Addison-Wesley
booth for copies of the CWEB book (what can I say -- I'm a pushy New
Yorker :-). I hope they're not disappointed (or overwhelmed).

\item[The presentation mentioned in the previous report]
Stephen Shum (from the CS Department of Augusta College) presented a
paper entitled ``Using Literate Programming to Teach Good Programming
Practice.'' His coauthor, Curtis Cook (from the CS Dept. of Oregon
State Univ), was not present.

What they wanted to determine was whether or not using LP techniques
lead to measurably better documentation. Unfortunately, they were
unable to come up with an experiment that could handle this, so they
settled (as a first pass; the project still continues) for trying to
determine if LP at least lead to *more* documentation, leaving the
evaluation of better or worse for later. (As those of us who teach
programming know, students tend to resist doing any documentation, so
``more'' can be seen as a necessary stepping stone to ``better.'')

The experiment was conducted in a single (upper division) course. [I
forget what the course was, and the paper doesn't mention it.]  An
```

```
(admittedly small) sample of 16 students (10 seniors, 3 juniors, and 3
``post college'') was split into 2 groups. Group A did the first
assignment using LP; group B the second.

While the number of comment lines was pretty constant between literate
and non-literate programs (a surprise), the literate programs used
significantly more words (and characters) in their comment lines. The
reason for this appears to be that LP comments are usually in the form
of paragraphs, which fill up a line, while ordinary program comments
tend to be short remarks sharing a line with a bit of code. When they
analyzed the content of the comments, they found that the comments in
the traditional programs were all of the ``what'' variety, while 5% of
the LP comments included ``how'' information and 1% of the LP comments
included examples.

From the paper:

        Although no inconsistencies were found between the internal
        comments and the code in both versions, we did find
        inconsistencies between the source code and external documentation
        in the traditional programs.

        Our experiment shows the literate programming paradigm indeed
        encourages more and consistent documentation. The students seem
        to like the literate programming style, except they do not like
        the debugging process required by the literate programming
        process.

The experiment used a ``home-grown'' LP system called AOPS. Like
CLiP, AOPS is programming language and typesetter independent. The
``markup language'' for AOPS seems much less intrusive than CLiP's,
however (sorry, Eric). On the other hand, it's difficult to see how
AOPS output would be typeset (there were no examples of ``woven'' output),
and Shum admitted that the students did not really typeset their
programs at all. It may be that AOPS is much better for tangling than
weaving. I'll be able to tell more when I get the copy of AOPS I was
promised.

\item[Another presentation related to LP]
On Friday afternoon, Philip Miller (from Carnegie Mellon) presented a
paper entitled ``Engaging Students and Teaching Modern Concepts:
Literate, Situated, Object-Oriented Programming'' (coauthored by Glenn
Meter, also from Carnegie Mellon). While the students involved in
the project described in the paper used a literate programming system
(another home-brew system called Genie, for the Macintosh), the LP
experience was only a side issue and was only dealt with in the following
statement (from the paper):

        We think the literate programming environment was very important,
        but we have not conducted any experiments to establish this as
        fact.

\end{list}
```

That's all from Phoenix (at least all I can remember). "There's no such thing as a foolproof system. And the more complex you make it, the more intricate, then the quicker things can happen if it ever breaks down." -- Peter George _Commander-1_ (1965)

Long array initializers in CWEB

From: **Peter Jensen**
Date: 10 Mar 1994

Here's a CWEB formatting hack I just figured out - maybe there's an easier way, but this works for me. The goal is to have an initialized array formatted roughtly as shown:

```
char stuff[size] =
{
  0,1,2,3,4,5,6,7,8,9,
  9,8,7,6,5,4,3,2,1,0
}
```

The problem is that there is nothing to trigger the line-break before the left brace, and the indentation within the braces. (I should have said, too, that I want to specify the line breaks within the list of values.) Here's how I did it:

```
char @[ stuff[size] = @] {
  @[ 0,1,2,3,4,5,6,7,8,9, @]@;
  @[ 9,8,7,6,5,4,3,2,1,0  @]@;
};
```

The @[...@] brackets coerce their contents to an "exp" (which the @; can then turn into a "stmt") and the right things happen.

Literary programming and C++

From: **Anssi Porttikivi**
Date: 15 Mar 1994

THE BACKGROUND. I have read the TeXbook and the LaTeX book through with medium concentration. I have used TeX a little. I tried to read "TeX - The Program" but it was too much for me. I have a copy of the original Knuth WEB report and have glanced through it. But I never had any real use for these methods before. Now we are considering CWEB for a middle sized four people programming project at The Helsinki University, Department of Computer Science. Our department uses LaTeX a lot but has very little experience with WEB. I have spent two days reading the literate programming stuff in WWW - an excellent proof of the tremendous effectiveness of Internet technologies of today as learning tools.

THE PROBLEM. However, I maybe stupid but I don't quite understand how to write a makefile and WEB source files to produce: 1) a single document with single contents and index page, and 2) lots of C++ header and implementation files. Maybe this is trivial and is done with CWEB @(object.H@> and @(object.C@> commands? How about multiple WEB source files? We are using all kinds of weird C++ features extensively, I don't think that could be a problem? And I am not sure if I am in the right track at all. And what do you think, should I complicate this even further by introducing SCCS to the project?

Good grief, tell me what is the fastest way for humble humans to write best possible code! With "best" I mean:

a) understandable

b) re-usable

c) with fool proof easy to use user interface

d) reliable

I don't give a damn about memory requirements or speed! I believe in systems simple enough to be obviously right, not in systems complex enough not to be obviously wrong. Why am I posting this anyway? I should be reading the docs and examples instead... And the whole project (program visualization, concentrating on string algorithms) should maybe be done in Visual Basic if we wanted to get it done fast...

From: **Thomas Herter**
Date: 16 Mar 1994

I wrote a lot of software using Knuth's original WEB and Pascal. I have done also a small project using CWEB and C. I have also done some experiments with MWEB, FWEB and Spider for ADA. I believe, that all the WEB's work similar and that any good programmer can adapt the Web-Approach within of two-three weeks of training.

From: **Andreas Stirnemann**
Date: 16 Mar 1994

I have been using C++ together with FWEB for about one year, and I am pleased with the results. Recently I switched to noweb, which I prefer to FWEB, mainly because of the ease with which it fits into the UNIX toolbox. I also find its syntax more flexible. My advice would be: Keep using C++, and use it together with noweb.

From: **Lee Wittenberg**

Date: 16 Mar 1994

Anssi.Porttikivi writes: I have read the TeXbook and the LaTeX book through with medium concentration. I have used TeX a little. I tried to read "TeX - The Program" but it was too much for me.

Me too. I browse through it occasionally, though.

However, I maybe stupid but I don't quite understand how to write a makefile and WEB source files to produce: 1) a single document with single contents and index page, and 2) lots of C++ header and implementation files. Maybe this is trivial and is done with CWEB @(object.H@> and @(object.C@> commands? How about multiple WEB source files?

Yes, that's what @(is for. If a number of different programmmers are working on different parts, then you will probably want to use @i a lot, too.

We are using all kinds of weird C++ features extensively, I don't think that could be a problem?

It may confuse the prettyprinter, but @[and @] can usually be used to unconfuse it. CTANGLE couldn't care less.

And I am not sure if I am in the right track at all. And what do you think, should I complicate this even further by introducing SCCS to the project?

Use SCCS, by all means. My serialno sample (in the pub/leew/LP.samples directory of bart.kean.edu) uses PVCS, but can show you a simple way of dealing with version control in a CWEB program. The palevent program (in the same directory) "goes whole hog" with respect to version control, but it's in noweb rather than CWEB. Feel free to use any ideas from these programs.

Good grief, tell me what is the fastest way for humble humans to write best possible code! With "best" I mean:

a) understandable

b) re-usable

c) with fool proof easy to use user interface

d) reliable

That's the philosopher's stone, isn't it? There's no way to guarantee any of that, but IMHO literate programming addresses every issue except c. I think most people on this list would agree.

I don't give a damn about memory requirements or speed!

"Premature optimization is the root of all evil." -- Knuth "Make it right before you make it faster." -- Kernighan & Plauger

I believe in systems simple enough to be obviously right, not in systems complex enough not to be obviously wrong.

Bravo! I wish Microsoft (and Borland and IBM and the rest of them) felt that way.

Why am I posting this anyway? I should be reading the docs and examples instead... And the whole project (program visualization, concentrating on string algorithms) should maybe be done in Visual Basic if we wanted to get it done fast...

Visual Programming systems lead to major maintenance problems in my experience, although using literate programming in conjunction with a VP system is possible (but difficult -- the VP systems usually don't like other tools muscling in on their territory).

Verbatim in CWEB?

From: **Denis Roegel**
Date: 22 Mar 1994

Has somebody devised a neat way of including some parts verbatim in a CWEB documentation? I want something like LaTeX's \verb. I tried to adapt it and put the following in the limbo:

```
\catcode`\*=11
\def\*makeother#1{\catcode`#1=12\relax}
\def\verb{\begingroup \catcode``=13 \*noligs
\tt \let\do\*makeother \dospecials\*sverb}
\def\*sverb#1{\def\*tempa ##1#1{\leavevmode\null##1\endgroup}%
\*tempa}
\begingroup
\catcode``=13
\gdef\*noligs{\let`\*lquote}
\endgroup
\def\*lquote{\leavevmode{\kern0pt}`}
\catcode`\*=12
```

(The main difference is that I replaced @ by *) This code works well with plain TeX, but not with CWEB. For instance, \verb/|---...---|/ does not give the right result. It is better in the C comments, but awful in the documentation. Being a little bit in a hurry, I don't have much time to investigate now, but maybe someone else has already worked this out ?

From: **Joachim Schrod**
Date: 22 Mar 1994

Denis Roegel writes: Has somebody devised a neat way of including some parts verbatim in a CWEB documentation? This code works well with plain TeX, but not with CWEB. For instance, \verb/|---...---|/ does not give the right result.

In particular, this example will not work easily. All stuff between vertical bars will be handled as `restricted C code material' by CWEAVE. Use |@t\verb/|---...---|/@>| . I know, it's ugly -- but don't ask me why @t is not allowed in documentation mode. (It was, in my CWEB :-) LaTeX users should note that cweb.sty does not support the above work-around, this is a known restriction. Otherwise your \verb macro looks functional. (Note: functional, not fine; your code is nearly unreadable. Please consider to use the modern method of `indentation', indispensable for coding any program, regardless of the language...)

From: **Denis Roegel**
Date: 22 Mar 1994

In particular, this example will not work easily. All stuff between vertical bars will be handled as `restricted C code material' by CWEAVE. Use |@t\verb/|---...---|/@>| . I know, it's ugly -- but don't ask me why @t is not allowed in documentation mode. (It was, in my CWEB :-)

It works perfectly. Thanks!

Otherwise your \verb macro looks functional. (Note: functional, not fine; your code is nearly unreadable. Please consider to use the modern method of `indentation', indispensable for coding any program, regardless of the language...)

Sorry to have posted this piece of ugly TeX code. The explanation is that I have taken the relevant excerpt of latex.tex and changed a few things without reformatting it.

From: **Marc van Leeuwen**
Date: 23 Mar 1994

Joachim Schrod writes: In particular, this example will not work easily. All stuff between vertical bars will be handled as `restricted C code material' by CWEAVE. Use |@t\verb/|---...---|/@>| . I know, it's ugly -- but don't ask me why @t is not allowed in documentation mode. (It was, in my CWEB :-)

Wow, that's a neat trick (or ugly, if you prefer)! It took me some time to figure out that this was just one `|...|' group, rather than two, and why. In fact I previously thought that ``It is impossible to have TeX text that contains the character `|' in any way (except in limbo), no matter what context one sets up (because CWEAVE acts before TeX)'' (quote from my manual), thinking only of TeX contexts (like `\verb/.../'), not of CWEB contexts (`@t...@>'). I found that in CWEB 3.1 you can even place an `@' in the text within `|@t\verb/.../@>|', as long as you don't forget to double it (this used to be forbidden in older versions of CWEB). Nonetheless the trick is quite fragile, and depends for instance on the fact that the dummy macro `\PB' that CWEAVE places around `|...|' groups is deactivated by \let\PB=\relax rather than by \def\PB#1{#1} (which whould freeze the catcodes before \verb has the chance to change them); if \PB is activated for some purpose, then the trick will therefore fail. And for the sake of completeness: the construction can be used within comments, but not within module names.

As to the question why `@t' is not allowed within TeX text, I think the answer is that Knuth argued that it could be of no use to insert a piece of TeX text when you are already in TeX mode; the counterexample you gave is contrived enough to be easily overlooked. It may also be noted that one can produce arbitrary output in typewriter type using the macro `\.' of the cwebmac format, although this is not a verbatim macro and some characters will need to be escaped; also to get a vertical bar one needs an indirect method, for instance defining \chardef\v=`| in limbo (it is unclear to me why this definition is not contained in cwebmac.tex) so that the above example could be written as `\.{\v---...---\v}'. This may be a bit more cumbersome, but the result is more robust.

Literate scripts in functional programming languages

From: **Andrew Butterfield**
Date: 23 Mar 1994

The WEB style of literate programming was created by D.E. Knuth during the development of his TeX typsetting software. All the original work revolves around a particular literate programming tool called WEB.

I am new to this group so I read the FAQ, some of which is quoted above. I notice (as an occasional literate programmer) that everything seems to revolve around WEB and variants. So I thought I'd let this group know that literate programming has spread it wings slightly further. A lot of functional programming languages have the notion of "literate scripts" where the usual comment convention is inverted. That is, all text is assumed to be comments, unless flagged otherwise. In particular program code can be embedded in code for a documentation system.

For example, Miranda literate scripts use '>' in column 1 to flag code lines, so the following fragment (indented for clarity) is BOTH correct Miranda AND correct LaTeX:

```
\documentstyle{article}
\begin{document}
We code up the factorial function directly
based on its common mathematical recursive definition:
\begin{displaymath}
n! = \left\{ \begin{array}{ll}
             0       & \mbox{if $n = 0$} \\
             n(n-1)! & \mbox{otherwise}
             \end{array}
     \right.
\end{displaymath}
This gives the following Miranda code:
\begin{verbatim}

> fac 0 = 1
> fac n = n * fac (n - 1)

\end{verbatim}
It is easy to see that it is correct.
\end{document}
```

The literate script approach is simpler than WEB, in that it won't re-arrange code to meet define-before-use rules, for example. However the languages that employ this usually don't have a define-before-use rule. Why did I post this - simply for information - WEB isn't the only class of tools supporting this kind of programming. I thought I'd let you know.

$64,000 question: are there any other tools (non-WEB, non-literate-scripts) for supporting literate code -- in particular does anyone use hypertext systems? P.S. Another commerical tool for supporting a formal specification methodology (IFAD Toolbox for VDM-SL) also has a similar idea, giving us: literate Specifications! (that'd be the day ! :-)

From: **Lee Wittenberg**
Date: 24 Mar 1994

Andrew Butterfield writes (in addition to a number of comments -- that I support wholeheartedly -- about literate programming not being restricted to WEB and its variants): A lot of functional programming languages have the notion of "literate scripts" where the usual comment convention is inverted. That is, all text is assumed to be comments, unless flagged otherwise. In particular program code can be embedded in code for a documentation system.*

For example, Miranda literate scripts use '>' in column 1 to flag code lines, so the following fragment (indented for clarity) is BOTH correct Miranda AND correct LaTeX: [example omitted for brevity] The literate script approach is simpler than WEB, in that it won't re-arrange code to meet define-before-use rules, for example. However the languages that employ this usually don't have a define-before-use rule.*

If by "simpler," you mean that it has fewer constructs that the programmer has to deal with, I agree. If, on the other hand, you mean that it is easier to use, I have to disagree. IMHO, tangling is the most useful part of literate programming: it allows me to develop my programs in a step-wise fashion, concentrating on small pieces at a time, without the necessity of rearranging these pieces for the convenience of the compiler (and the attendant errors that occur in this process).

While getting around the define-before-use rule is one use of tangling, it is useful in many other contexts. The following example (taken from the Stanford GraphBase) will illustrate:

```
@<The |risc| routine@>=
Graph *risc(regs)
  unsigned long regs;       /* number of registers supported */
{
  @<Local variables for |risc|@>@;@#

  @<Initialize |new_graph| to an empty graph of the appropriate size@>;
  @<Add the RISC data to |new_graph|@>;
  if (gb_trouble_code) {
    gb_recycle(new_graph);
    panic(alloc_fault);     /* oops, we ran out of memory
                               somewhere back there */
  }
  return new_graph;
}
```

In this case, the chunks (except for @<Local variables...@>) are used for code refinements rather than declaration order. Granted, function calls could be used to accomplish the same purpose, but CWEB chunks carry no run-time overhead (and I find chunk names more informative than function names -- I despise long, run-on function names, but chunk names are naturally in the form of full sentences).

Why did I post this - simply for information - WEB isn't the only class of tools supporting this kind of programming. I thought I'd let you know.

The information *is* appreciated (that's what the newsgroup/list is for). I am not familiar with Miranda, myself, and was therefore not aware of this (extremely worthwhile) feature. It reminds me of a language in the 60's (I forget its name, but it's described in Sammet's Programming Languages: History and Fundamentals) where comments were typed with a red ribbon and code with a black one (or vice versa), but otherwise freely interspersed.

$64,000 question: are there any other tools (non-WEB, non-literate-scripts) for supporting literate code -- in particular does anyone use hypertext systems?

A number of contributors have expressed interest in such systems in the past; I assume that at least one of them is working on something. Anyone?

From: **Joachim Schrod**
Date: 24 Mar 1994

Andrew Butterfield writes: I am new to this group so I read the FAQ, some of which is quoted above. I notice (as an occasional literate programmer) that everything seems to revolve around WEB and variants. So I thought I'd let this group know that literate programming has spread it wings slightly further. A lot of functional programming languages have the notion of "literate scripts" [...] The literate*

script approach is simpler than WEB, in that it won't re-arrange code to meet define-before-use rules, for example.

Thanks for your article. There has been lots of discussions on the literate programming mailing list (that eventually evolved into a newsgroup) about that point. The point most discussant agreed on in the end was that the support of refinements (i.e., to bind code chunks to names and be able to use the names in place of the code itself) is a crucial part of every literate programming system. (You may also want to check the discussions in the CACM column about this topic.)

Systems that support typeset documentation are therefore usually not considered to be literate programming systems. I would call them literate documentation systems. They are a very valuable contribution as a handy tool (I have written some of them myself ;-), but they are not the real beef. Btw, that's the reason why the regular posting on the contents of the LitProg Archive lists such systems in a separate section. I think I'll add the name LitDoc system to the title of this section. :-)

However the languages that employ this usually don't have a define-before-use rule.

That's not the point of refinements. It's a nice by-product to circumvent this rule in languages that demand them, but it's not necessary. Refinements allow the expression of an abstraction, like functions do. But as an author I can impart this abstraction much better since I can use real text to give it a title, instead of a simple identifier. [I hope `impart' is the right verb here; I don't have a dictionary handy...] Btw, there is also a technical difference between refinements and functions: Refinements are still in the same lexical closure, from a pure hacker's point of view they are just macros. But I find this view too shortsightened, it does not acknowledge the importance of presenting one's abstractions to a reader of one's code. So refinements still have their value in functional languages (and OO languages, for that matter; where one could argue that methods don't need refinements with similar arguments). Anytime I write a larger CLOS program I'm missing a literate programming environment for it... ;-)

PS: I tried to recapitulate the previous discussions as best as I could. Feel free to correct me, as many of you know I'm very biased. But perhaps one should mention the discussion in the FAQ?

Refinements

From: **John Ramsdell**
Date: 24 Mar 1994

Joachim Schrod writes: Thanks for your article. There has been lots of discussions on the LitProg mailing list (that eventually evolved into a newsgroup) about that point. The point most discussant agreed on in the end was that the support of refinements (i.e., to bind code chunks to names and be able to use the names in place of the code itself) is a crucial part of every LitProg system. (You may also want to check the discussions in the CACM column about this topic.)

Systems that support typeset documentation are therefore usually not considered to be LitProg systems. I would call them Literate Documentation (LitDoc for short) systems. They are a very valuable contribution as a handy tool (I have written some of them myself ;-), but they are not the real beef. Btw, that's the reason why the regular posting on the Contents of the LitProg Archiv lists such systems in a separate section. I think I'll add the name LitDoc system to the title of this section. :-)

I think this note reveals a source of confusion about the notion of a literate programming system. Let me expose the problem by means of an example. The SchemeWEB filter provides simple support for literate programming in Lisp. The filter does not provide support for refinements, so by the logic above, one might say that it only supports literate documentation. However, SchemeWEB was created to be used with Scheme with provides an extremely powerful refinement mechanism called hygienic macros. This mechanism gives macros writers control over the scope of macro parameters so that they can avoid inadvertent name capture. This and other features make it much more useful than ordinary textual macro expansion used in WEB and its relatives. If you define your system to be the combination of SchemeWEB and a Scheme implementation with hygienic macros, then it is easy to see that the combination is a literate programming system. The down side of the use of SchemeWEB is the lack of automated support for the generation of indices, but that is a different subject. A tool's inclusion in the section titled literate documentation systems just means the tool itself does not do refinements; it does not tell you whether the tool supports literate programming.

From: **Joachim Schrod**
Date: 25 Mar 1994

John Ramsdell writes: I think this note reveals a source of confusion about the notion of a literate programming system.

I wouldn't call it confusion -- I would call it a serious difference made consciously.

[Scheme macros] ... more useful than ordinary textual macro expansion used in WEB ...

As long as one sees refinements *only* as a macro expansion, than you're right. But -- as I tried to explain in the part of my article you discarded -- refinements are more. I *cannot* use long verbal sentences, with TeX formatted formulas in them (for pre- and postconditions!), as macro names. And identifiers are no substitutions for that. IMO the Scheme (or CL) macro mechanism supports another area of abstractions: With it one can define new syntactic constructs to increase the abstraction level of the formal part. That's something completely different for me. (But I use it a lot, actually!)

IMNSHO, a similar line of reasoning could be made to drop loops from programming languages because we have condition and gotos. It's there, yes -- but this is a different *abstraction* level. To express the reason of one's abstractions with prosa is better than with some identifiers -- that's the heart of Literate Programming! Why do we document then, after all; good programs with well-chosen identifiers and very small functions or methods, invariants etc. listed in comments, are self-documenting and everything we need, aren't they? (At least Dijkstra thinks so... ;-)

Don't get me wrong, the comments above are in no way meant personally. I simply want to show why refinements are important for me; I wanted to make the points you deleted more clearly. (Obviously I wasn't good enough in formulating them.)

From: **John Ramsdell**
Date: 25 Mar 1994

Joachim, I find it very interesting to learn about the very deeply thought out opinions you have about refinement, and I hope you find hygienic macros interesting due to their ability to give programmers more control over macro expansion. However, I find it hard to believe most people will appreciate these distinctions. I realize that my previous posting might be construed as lending implicit support to the notion that literate programming tools must support refinement. I see no reason to be so restrictive. When asked, here is how I describe literate programming:

Literate programming is a style of programming in which programmers view their task as communicating computational processes mostly for the benefit of other humans, rather than solely for the benefit of computing machines. A literate programming tool simply supports this style of programming. In particular, I think that the paper and pencil that Donald Knuth used to write the first draft of TeX are literate programming tools. In my opinion, this news group should be dedicated to promoting the style of literate programming in whatever form it may take.

John Ramsdell writes: I think this note reveals a source of confusion about the notion of a literate programming system.

Joachim Schrod writes: I wouldn't call it confusion -- I would call it a serious difference made consciously.

Sorry, I made a bad choice of words. Your characterization is more accurate.

From: **Christian Lynbech**
Date: 29 Mar 1994

At the danger of mudling up this discussions, I will offer my own version of why something like scheme's macro facility isn't strictly Literate Programming to me either. To me, the refinement aspect of literate programming is the most important. This sets literate programming (or rather the tools that support it) aside from various hacks one could do for modularizing and formatting ones code.

Refinements are important because they support one of the two dimensions of Structure Programming, which I believe like many others, is a good thing (I'll come back to these dimensions). Knuth says in his article on programming with goto's (included in his book: Literate Programming): "We understand complex things by systematically breaking them into successively simpler parts and understanding how these parts fit together locally." Or as Tony Hoare said, when Knuth asked him to define structured programming: "The systematic use of abstraction to control a mass of detail, and also a means of documentation which aids program design."

The refinement notion has a definite advantage in being low-cost (as oposed to real functions or procedures), syntax-independent and transparent to the product. Language macros are intricate parts of the product, and may have subtile effects not easily recognized from the code alone, i.e. they may need real debugging. The concepts of purely syntactically expansion is much easier. As it has already been said: macros is a device for extending the syntax of the language.

The two dimensions of structured programming, is something I read somewhere I unfortunately cannot remember. But if I remember correctly, the main idea is that structured programming has two dimensions: one horizontal and one vertical. The horizontal covers modularization of the code, with clearly defined interfaces between the components. This is supported by the language and its type system. For instance, pascal has rich constructs for making a good job here, with procedures and composite types. The vertical dimension covers the stepwise construction of the program, working from the large perspective towards the fine details, something elegantly supported by refinements. As I see it, few languages support this dimension to any reasonable degree. Scheme perhaps,

but certainly not neither Pascal nor miranda. Their function/procedure concept is much too coarse-grained for this purpose.

From: **John Ramsdell**
Date: 30 Mar 1994

I think that calling the macro expansion activity employed by many literate programming tools refinement is very misleading. To me, refinement conjures up some notion of preserving some aspect of the semantics of an object while adding in more semantics in the form of making the object more concrete. The most complex literate programming tools provide an abstraction mechanism implemented by the most primitive form of macro expansion: character based expansion in which the strings are uninterpreted.

Tools that really support refinement have some knowledge of the semantics of the object being refined. For example, consider Kestrel's REFINE tool. It manipulates representations of algorithms that have been assigned a formal interpretation in a precisely defined logic. You might read Richard K. Jullig's article "Apply Formal Software Synthesis" in IEEE Software May 1993, so see how others use the term refinement.

It is okay to say that many literate programming tools provides primitive support for refinement, but let's be clear as to how minimal that support is. These tools facilitate the documentation of the refinement processes. They really are refinement documentation tools. In my opinion, focusing on refinement simply emphasizes a short coming of the current generation of literate programming tools.

From: **Joachim Schrod**
Date: 30 Mar 1994

John Ramsdell writes: I think that calling the macro expansion activity employed by many literate programming tools refinement is very misleading. [...] Tools that really support refinement have some knowledge of the semantics of the object being refined. For example, consider Kestrel's REFINE tool.

I have to confess that I use the term `refinement' as it was coined by N.Wirth in "Program Development by Stepwise Refinement" [CACM 14(4), p.221-227] and used by Dijkstra & Hoare in _Structured Programming_ and by Gries in _The Science of Programming_. There refinements don't "have some knowledge of the semantics of the object being refined" per se; there are pre- and postconditions both the refinement placeholder and the refinement itself obey, the refinement often more strictly. See also Hoare's new article "Algebra and Models" [SEN 18(5), p.1-8], an abstract of his keynote address to the 1st ACM Symposium on the Foundations of Software Engineering. Up to now this usage was sufficient for me (as I don't need _more_ formalism than Dijkstra uses...)

You might read Richard K. Jullig's article "Apply Formal Software Synthesis" in IEEE Software May 1993, so see how others use the term refinement.

Oh yes, I remember that paper. There `refinement' was used for the semi-automatic transformation of an abstract data type to a concrete data type (by an inference machine based on a description in 1st order logic). It might be that this is the usage in the AI field (where Jullig comes from obviously), but this is definitely _not_ so in software engineering. Compare, e.g., to Liskov & Guttag's _Abstraction and Specification in Program Development_ (and for me Liskov is definitely an authority in this area, after all _she_ coined the term `abstract data type'!). Or Ian Summerville's _Software Engineering_. (Btw, Ian has also written a good article about the problems between SE and AI folks; may be fetched by anonymous ftp from ftp.comp.lancs.ac.uk.)

To me, refinement conjures up some notion of preserving some aspect of the semantics of an object while adding in more semantics in the form of making the object more concrete.

Yes, here I agree with you. And I don't see the contradiction: A piece of code is more concrete than a placeholder that notes the pre- and post-conditions. As explained by Wirth in his article that is worth to be read from time to time...

From: **Marc van Leeuwen**
Date: 31 Mar 1994

Joachim Schrod writes: Yes, here I agree with you. And I don't see the contradiction: A piece of code is more concrete than a placeholder that notes the pre- and post-conditions.

Sorry about my ignorance on this point but what is all this talk about pre- and post-conditions about? Has anyone ever seen a real (as opposed to toy) pre- or post-condition? Could you mention a module name in TeX The Program or the Stanford GraphBase (or some other published literate program) that mentions such a condition? It's probably due to my wrong upbringing, but to me pre- and post-conditions always seemed to be extremely impractical formal expressions to be attached to parts of imperative programs (often

larger than the part itself) for proving that the program meets a formal specification (and which some would believe are helpful in finding an implementation for those specifications); something that I have never seen done for a real-world program. This is not a criticism of the remarks above, just a question to enlighten me about the true meaning of these terms.

From: **Lee Wittenberg**
Date: 31 Mar 1994

Marc van Leeuwen writes: Sorry about my ignorance on this point but what is all this talk about pre- and post-conditions about? Has anyone ever seen a real (as opposed to toy) pre- or post-condition? Could you mention a module name in TeX The Program or the Stanford GraphBase (or some other published literate program) that mentions such a condition? It's probably due to my wrong upbringing, but to me pre- and post-conditions always seemed to be extremely impractical formal expressions to be attached to parts of imperative programs (often larger than the part itself) for proving that the program meets a formal specification (and which some would believe are helpful in finding an implementation for those specifications); something that I have never seen done for a real-world program. This is not a criticism of the remarks above, just a question to enlighten me about the true meaning of these terms.

I can't speak for literate programs, but the programming language Eiffel provides support for "real" pre- and postconditions that are extremely practical. Meyer describes their use as "programming by contract," an extremely powerful idea and the best (some would say only, but I'm not about to get involved in *that* religous war) reason to program in Eiffel. Meyer describes the concept (and the language) in a number of books and articles, most notably _Object-oriented_Software_Construction_ and _Eiffel:_The_Language_.

From: **Joachim Schrod**
Date: 05 Apr 1994

Joachim Schrod writes: Yes, here I agree with you. And I don't see the contradiction: A piece of code is more concrete than a placeholder that notes the pre- and post-conditions.

Marc van Leeuwen writes: Sorry about my ignorance on this point but what is all this talk about pre- and post-conditions about? Has anyone ever seen a real (as opposed to toy) pre- or post-condition?

Yes, in the interface specifications of my modules. Actually, I don't use _formal_ conditions very often. Sometimes, they tend to be as complex as the programs themselves and they don't raise the abstraction level necessarily. In particular, formal conditions for user interface modules are notoriously difficult to formulate; and that's where my current work focuses on. Nevertheless semi-formal conditions are very helpful. With `semi-formal' I mean that they are mandatory; they have to use a kind of prose that explains the changes in state, i.e., they have to be verbal expressions of predicates that hold before and/or after the activation of that interface.

It is also of interest to preclude this kind of specification with a description of the general model one has in mind of the abstraction a component is responsible to represent. If one's analysis and design method is data-driven, it's useful to use prose that orients on the equations and restrictions formalism of algebraic specifications (one can neglect errors and exceptions since they are covered in the interface specification part). In connection with abstract functions and representation invariants this provides very good documentation on the basic assumptions one has used in the contract (ie, the specs) and the implementation. Of course, for critical modules formal specs might be in order, too. Personally, I prefer the Larch framework in that case.

For a longer discussion on the background of this posting you might want to read

```
@book{spec:liskov:abstraction,
 author = {Barbara H. Liskov and John V. Guttag},
 title = {Abstraction and Specification in Program Development},
 publisher = mit,
 year = 1986,
 isbn = {0-262-12112-3},
 library = {D.1/Lis},
 annote = {\js{} Well written introductionary book on abstraction
    entities and specification. Uses CLU and Larch (ie, LSL \& Larch/CLU)
    for formal expressions.}
}
@string{ mit    = "MIT Press"}              % isbn: 0-262
```

I can only recommend this book, it features one of best discussions about the pros and cons of formal specification and presents the ideas on `semi-formal' specifications I outlined above. (Well, but I'm biased since I consider Liskov's work on abstractions and Guttag's work on formal specifications among the most important ones in this area.) Concerning the idea of distinguishing the specification of the abstract model of a component and its interface; this was presented first in the two-tier model of Larch. A good overview may be found in

```
@inproceedings{spec:guttag:two-tiered,
 author = {John V. Guttag and Jim J. Horning},
 title = {Formal Specifications as a Design Tool},
 booktitle = {Proceedings of the 7th ACM Symposium on Principles of
    Programming Languages},
 address = {Las Vegas, NV},
 month = jan,
 year = 1980,
 pages = {251-261},
 organization = acm,
 publisher = sig-plan,
 note = {Published as } # sigplan # { 15(??), 1980
 \unskip. {\bf SIGPLAN number is missing}
},
 annote = {\js{} Presents the first idea on the ``two-tiered approach,''
    which became later the basic principle of the Larch project. Note that
    it was Jeannette Wing who really worked out the whole paradigm.}
}

@article{spec:guttag:larch,
 author = {John V. Guttag and Jim J. Horning and Jeannette M. Wing},
 title = {The {Larch} Family of Specification Languages},
 journal = ieee-sw,
 volume = 2,
 number = 5,
 month = oct,
 year = 1985,
 pages = {24-36},
 library = {HeLaHoBi Zb 7098},
 annote = {\js{} Special section on the two-tiered approach and on
the specification language Larch. These are the basic papers.}
}
@string{ ieee-sw   = "IEEE Software"}
```

There is also a new book on Larch that I haven't read yet:

```
@book{spec:horning:larch-book,
 author = {Jim J. Horning and John V. Guttag},
 title = {Larch: Languages and Tools for Formal Specification},
 publisher = springer,
 year = 1993,
 isbn = {0-387-94006-5},
 library = {},
 annote = {}
}
```

Seeking account of experiences with noweb

From: **Norman Ramsey**
Date: 31 Mar 1994

If you've used noweb on a project that mattered to you (only you know what's important), I'd like to talk to you. I'm publishing an article about noweb, and the editorial staff of IEEE Software would like to hear more about how people (other than me and my colleagues) have used noweb. They're especially interested to hear about using noweb in ``practical'' situations. (Again, if you think it was practical, then it was.) If you're in North America, send me your phone number and I'll give you a call. If you're elsewhere, email may be easier than a phone call. Just for fun, I'd also like to hear from nuweb people, since nuweb and noweb are as alike as two peas.

From: **Dave Thompson**
Date: 01 Apr 1994

If you've used noweb on a project that mattered to you (only you know what's important), I'd like to talk to you. I'm publishing an article about noweb, and the editorial staff of IEEE Software would like to hear more about how people (other than me and my colleagues) have used noweb. They're especially interested to hear about using noweb in ``practical'' situations. (Again, if you think it was

practical, then it was.)

I've used noweb on a project just recently. It is a FORTRAN (don't laugh) program to read meteorological data from a couple of files and compute evapotranspiration using FAO-24 Penman. The programs comprise somewhere between 1000-2000 lines; the documentation (woven) is over 50 pages.

I learned a lot about literate programming with this experience. On review, I think that I have some conceptual errors in my web because I was forced to study the tangled code to find some not-too-subtle errors in the algorithms. Of course (although if pinned I'll deny all knowledge of the following statement ;-), I was forced to resort to my rather archaic and traditional debugging technique of using write statements to determine some errors in the code. <grin> I'll have to produce another module for this program to read data from a different format file and recompute evapotranspiration values for a longer period-of-record for the project. This will probably be simple because all of the hard work is already done.

For the record, I'm no professional programmer---I'm an engineer and researcher. My subjective evaluation is that I produce (at least) better documented code using literate techniques. The woven output will eventually be published as a Water Resources Center technical report. I'll probably do some cleanup prior to publishing, but the essentials will remain unchanged.

Hope this helps...call me if you like at (806) 742-3485. I've also used nuweb to a more limited extent. I like that ability to concatanate chunks so I can develop the code and logic simultaneously; nuweb doesn't provide for this (at least that I can tell).

From: **Simon Wright**
Date: 03 Apr 1994

Dave Thompson writes: Hope this helps...call me if you like at (806) 742-3485. I've also used nuweb to a more limited extent. I like that ability to concatanate chunks so I can develop the code and logic simultaneously; nuweb doesn't provide for this (at least that I can tell).

The nuweb I have here certainly seems to do this, on the simple test I've just done.

I've written a couple of programs in CWEB (a newbie here), but like and need the ability to handle multiple program & language files in the one web. The thing that bothers me, in prospect, is the interaction between nuweb & make; it's not practical to make the tangled output files depend explicitly on the web file, I think. I suppose the technique used by nuweb itself is the answer here?

```
all:
        $(MAKE) $(TARGET).tex
        $(MAKE) $(TARGET)
```

Information on literate programming?

From: **Greg Fiehler**
Date: 06 Apr 1994

Greetings, I was introduced to literate programming late last year and have been attempting to learn CWEB and literate programming practices since then. First I would like to thank everyone here that has helped me out by answering my past questions.

I'm a undergraduate at University of Missouri St. Louis and my advisor would like me to address our ACM student chapter on literate programming to try and generate some interest in its use on our campus. I have myself just started to realize the potential of using these methods and am convinced that they are worth pursuing. But I feel that besides talking about the advantages that I've seen in my own limited work and those that I have read about in Knuth's Literate Programming book I would also like to tell my fellow students that these methods and tools are actually being used in the programming community and not just as an acedemic exercise. I see the potential that these methods could have but I also think it will be difficult to convince a group to learn these methods in order to use them in a large project. If anyone can give me some examples of actual usage of literate programming or how you convinced a group to utilize these methods I would appreciate it very much.

Also since I am still new at this if anyone has some "top 10" points I should mention in a talk on literate programming or any other ideas to help generate interest I would also appreciate this. As a personal side I was wondering what other schools are doing to promote the use of literate programming tools and practices.

From: **Mary Bos**
Date: 06 Apr 1994

I'm in a graduate software engineering program at Seattle University in Seattle Washington. One of my classes looked at literate programming as a curosity (I presented a preselected paper that presented an algorithm in literate programming). I happen to be interested in literate programming but my classmates mostly pooh-poohed the idea. On the otherhand, I introduced literate programming where I worked. And now there is an internal tools group developing software using literate programming and they believe they are producing higher quality work (an internal paper will be presented on this at an internal conference). This group consists of about 3 or 4 people working in several programming languages. If the paper is released for public consumption, I'll post the reference.

Are there WEBs for visual languages (Visual Basic)?

From: **Stephen Boyan**
Date: 19 Apr 1994

Is there any work being done about webbing for languages like Visual Basic?

From: **Lee Wittenberg**
Date: 20 Apr 1994

I've done some literate programming work in Paradox for Windows, which is also a visual programming environment, but I haven't done anything in Visual Basic. The problem with PfW (and, I suspect, with VB) is that it isn't very tolerant of outside tools. I ended up using noweb to create text files that I then imported into the text windows for attaching ``methods'' to ``objects'' (it was really attaching procedures to widgets). It was a bit of a hassle, but I found it to be several orders of magnitude more pleasant than working directly in PfW.

If VB is a system where you ``drag & drop'' to create a screen interface and attach procedures to the widgets in the interface, you might find noweb helpful. I'll be glad to send you the details if you're interested. Write me directly, as there's no point boring the rest of the group :-).

From: **Weiqi Gao**
Date: 20 Apr 1994

I don't know, and I don't think anybody mentioned it is being done. So I assume it is not being done. I seriously considered modifying one of the webs for use with windows programming (Borland C/C++, or Visual C++), but the number of keywords kept me from doing it (take a look at windows.h). The DOS/Windows environment simply don't have the memory required to handle it. I guess the Visual Basic situation is the same.

Suggestion for function prototypes in CWEB

From: **Phil Jensen**
Date: 21 Apr 1994

Hello literate programmers, or more specifically CWEB users: I have implemented a small enhancement to CWEB on which I solicit comments. IMHO the necessity to predeclare functions is one of the misfeatures of C (compared to a `real' language like, say, Modula-3 ... but that's another subject). The inevitable sections that run:

```
@ @<Predecl...@>=
void silly_function(void);  /* Hey compiler!  Not an |int| function! */
```

are simply an annoyance. Therefore, what do y'all think about the following lightweight scheme to handle this. (BTW, you must be using ANSI C, i.e., with prototypes, for this...) Precede all function declarations with an extra control code `@~', thus:

```
@~double sqrt(double N)
{
  ... and so on
```

At one place in the w file, use the control code `@n'; usually right after @<Global variables@>.

The function headers are collected and duplicated at the point where the @n occurs (and ctangle could be made to say <Function

prototypes> just the way it now says <Preprocessor definitions>). The @~ construct does not need a closing delimiter, because the function header ends with the first un-nested right parenthesis. At the point where the prototypes are emitted, of course, each header is followed by <semicolon><newline>. [I know that one might come up with pathological return types that use parentheses; but it seems to me that the simplicity of this scheme is preferable: you can always use typedefs to avoid this pitfall.] If anyone wants to play around with this in practice, I can send the .ch file.

From: **Preston Briggs**
Date: 22 Apr 1994

IMHO the necessity to predeclare functions is one of the misfeatures of C (compared to a `real' language like, say, Modula-3 ... but that's another subject). The inevitable sections that run:

```
@ @<Predecl...@>=
void silly_function(void);  /* Hey compiler!  Not an |int| function! */
```

are simply an annoyance. Therefore, what do y'all think about the following lightweight scheme to handle this. (BTW, you must be using ANSI C, i.e., with prototypes, for this...) Precede all function declarations with an extra control code `@~', thus:

```
@~double sqrt(double N)
{
  ... and so on
```

At one place in the w file, use the control code `@n'; usually right after @<Global variables@>.

The way I usually handle these things is to collect all the prototypes in an appendix at the end of the document. This gives me a place to describe, cleanly, the interface for each routine, with types and means of each parameters. Inside the appendix, the prototypes for related groups of rotuines can be grouped together into subsections (sort of like the methods of a class). Overall, the hope is that it'll turn out something like another index that the reader or maintainer can refer to for quick answers. Of course, if your particular language doesn't require prototypes (or other interface specifications), you might still do this, as a matter of style.

From: **Stuart Ferguson**
Date: 25 Apr 1994

IMHO the necessity to predeclare functions is one of the misfeatures of C (compared to a `real' language like, say, Modula-3 ... but that's another subject). The inevitable sections that run:

```
@ @<Predecl...@>=
void silly_function(void);  /* Hey compiler!  Not an |int| function! */
```

are simply an annoyance. Therefore, what do y'all think about the following [resonable suggestion deleted]

I too have found this a problem, and since I use a language-independent web, I can't use your extension. There is a problem with this extension as well in that some functions might be global and need to be in the external header file and some might be private, whereas your scheme only provides a single blob of prototypes.

I have developed a style to deal with this problem. For the types of code I am writing, which is mostly libraries of tools, I break my document into two parts. The first part is an interface description which includes types and function declarations that the user of the library will need, and the second part is the implemetation which contains the actual code. I can usually just deliver the first part to my users and it acts as a reference. In the first part I define things of the type:

```
<< ExternalEntryPoint usage >> =
void
ExternalEntryPoint (
        int arg1,
        Thingee arg2)
@
```

I embed a bunch of these in appropriate text along with some << Public Type >> declarations and I have the user guide and an interface description from which to derive the implementational code. In the second part, the "usage" tags occur in two places:

```
<< Public Functions >> =
        << ExternalEntryPoint usage >>
        {
                << implementation code body >>
```

```
        }
@
```

And:

```
<< Public Prototypes >> =
        extern << ExternalEntryPoint usage >>;
@
```

I dislike having to write the prototype statement for each entry point, but once I do it the type for the function is written in one place and can easily be changed.

For private functions which are present in a single module and not used outside, I try to organize them into an appropriate order in the source file so that they are defined before they are used. Three classes are usually enough. "Public Functions" (as shown above) come last since they all have prototypes. "Private Functions" come before them since they will be used by the public functions, and "Private Utilities" comes first. Public functions can use all types of functions; private functions can use public functions and private utilities; private utilities cannot use either of the other function types. With the exception of loops, this almost always gives me enough distinctions to have all functions defined before they are used. For loops -- A uses B and B uses A -- I code the prototypes manually.

How do I begin literate programming?

From: **Eric Lemings**
Date: 22 Apr 1994

The FAQ did not answer my question. I don't have the books mentioned in the FAQ but I'd like to proceed without reading them first. I use a DOS system, C and C++ languages and I do not have any TeX or LaTeX tools. Which tools are the best? Which tools do I need? I would also like to be able to install the neccessary tools on an RS/6000 running AIX 3.2. Thanks in advance for any help.

From: **Tony Coates**
Date: 22 Apr 1994

Eric Lemings writes: The FAQ did not answer my question. I don't have the books mentioned in the FAQ but I'd like to proceed without reading them first. I use a DOS system, C and C++ languages and I do not have any TeX or LaTeX tools. Which tools are the best? Which tools do I need? I would also like to be able to install the neccessary tools on an RS/6000 running AIX 3.2. Thanks in advance for any help.

Oh boy, this is always fun when someone asks which literate programming tool is *best*. It depends a bit on what you do, and what you want. Of course, when you start, you don't know what you want, which is a difficulty, but anyway, here are some comments:

(i) Literate programming tools can generally be broken up into two groups - those which are aware of language grammar, and those which aren't. CWEB (for C/C++) and FWEB (for C,Fortran,others) are perhaps the most established grammar-aware tools. They can provide a degree of program checking before you reach the compiler. I have never used one of these tools. I prefer to use a language-insensitive tool, because I have documents which generate different types of files, e.g. C++ code and matching makefile code, and I prefer to have a tool which isn't going to be confused by different and arbitrary file types. Language insensitive tools include FunnelWeb (which I use), noweb, and nuweb (the latter two both have many followers). The language-independent tools are often considered to be simpler to use, but can't do clever formatting such as highlighting keywords. This doesn't matter to me, but it may to you. If you really do just want to work on C/C++ code, for instance, you might find the investment in using CWEB/FWEB worthwhile. If, like me, you find youself writing documents containing mixed code (e.g. for me Maple, C++, makefile, UNIX and OS/2 shell scripts) in the one file, then a language-independent file may be useful.

(ii) Most tools output TeX/LaTeX, so it would be to your advantage to have one or the other installed. CLiP and noweb can generate HTML hypertext (which can be viewed with Mosaic) (I'm adding this to FunnelWeb at the moment, though note that I am not the author). However, for printed output, most tools support TeX/LaTeX because there aren't many other established text standards yet (SGML is yet to catch on enough, RTF (Microsoft) is a moving target). There is a simple tool, WinWordWeb, which works within Microsoft Word, if this is of interest.

(iii) As to porting to the RS6000, I'm not aware which tools if any have been ported to the RS6000. Having ported FunnelWeb onto several platforms now, I can state that this tool is written with portability very much in mind (e.g. moving from a 32-bit to a 64-bit architecture involved little more than changing one definition in a header file; I wish my other programs would port so easily). If you wanted to port FunnelWeb to the RS6000, I'd be quite happy to provide any help that I can, and to integrate any necessary changes into my current development version (which I hope will eventually be integrated with the next official release of FunnelWeb). Currently, FunnelWeb runs under System 7, DOS 5.0, OS/2, SunOS, VMS, and OSF/1. Adding AIX would probably not be too hard. Anyway, I hope that some of this helps (excuse my FunnelWeb bias folks, but it is what I use every day in my work).

From: **Lee Wittenberg**
Date: 23 Apr 1994

Eric Lemings asks: The FAQ did not answer my question. I don't have the books mentioned in the FAQ but I'd like to proceed without reading them first. I use a DOS system, C and C++ languages and I do not have any TeX or LaTeX tools. Which tools are the best? Which tools do I need? I would also like to be able to install the neccessary tools on an RS/6000 running AIX 3.2. Thanks in advance for any help.

Your question is likely to start a religious war, but I will do my best to provide a non-sectarian answer.

As to TeX, there are a number of good DOS versions around. My personal favorite is Eberhard Mattes's emTeX, because it is a high quality implementation (I've had no problems with it, and I'm a bug magnet) and it's freeware (I'm cheap). emTeX, and most of the other DOS implementations, are available from the CTAN archives (address in the FAQ). [Note: The LaTeX macro package is usually included with a TeX distribution -- it is definitely included with emTeX. On the other hand, LaTeX is also available from the CTAN archive.]

As to literate programming tools, I shall limit myself to listing the tools that I know will work under DOS for C/C++ programs. The location of all these tools is in the FAQ.

1. Tools I use (under DOS)

CWEB

noweb

2. Tools I know (sic) work under DOS

CLiP

Nuweb

FunnelWeb

FWEB

WinWordWEB

WEB system for Perl or Modula-2

From: **Gregory Tucker-Kellogg**
Date: 22 Apr 1994

I am new to the uses of literate programming. Has anyone built a WEB system for Perl? I would be interested in such a beast. If not: while I know that there are several language-independant WEB implementations, I don't know how popular or widely used they are. It's hard to get a sense from the FAQ about what people really prefer. I would like to get such a sense before compiling up a bunch of webs on my machine.

From: **Lee Wittenberg**
Date: 23 Apr 1994

Gregory Tucker-Kellogg writes: I am new to the uses of literate programming. Has anyone built a WEB system for Perl? I would be interested in such a beast. If not: while I know that there are several language-independant WEB implementations, I don't know how popular or widely used they are. It's hard to get a sense from the FAQ about what people really prefer. I would like to get such a sense before compiling up a bunch of webs on my machine.

I think you're probably better off using a language-independent system. I don't think that popularity is quite as important as support and comfort (yours, not the tool's). To the best of my knowledge, CLiP, FunnelWeb, noweb, and Nuweb (in alphabetical order) are all well-supported and available. Just pick the one whose ``look and feel'' you prefer. My preference is noweb, but 9 out of 10 doctors recommend sugarless gum :-).

From: **Preston Briggs**
Date: 23 Apr 1994

Gregory Tucker-Kellogg writes: While I know that there are several language-independant WEB implementations, I don't know how popular or widely used they are.

Many people use noweb, nuweb, and FunnelWeb. They're all language independent and should suit your purposes. Nuweb and noweb are slightly more refined, I think, offering nicer crossreferences and indexing. Noweb is more flexibly structured than the others (they're monolithic programs; noweb is a collection of smaller programs).

From: **Phil Bowman**
Date: 01 May 1994

Is there a version of WEB suitable for programming with Modula-2 rather than PASCAL? If not, which of the non-language-specific WEBs would work with it? Does anyone have any experience of Web/M2 programming? (I have only just discovered the comp.programming.literate newsgroup)

From: **Preston Briggs**
Date: 01 May 1994

Phil Bowman writes: Is there a version of WEB suitable for programming with Modula-2 rather than PASCAL? If not, which of the non-language-specific WEBs would work with it?

The popular language-independent systems (FunnelWeb, noweb, nuweb) will all handle Modula-2. I'm biased toward nuweb.

Macro/scrap definition within a context?

From: **Mike Elliott**
Date: 29 Apr 1994

In working with languages such as Ada and Modula-3 which already have good encapsulation, I find a need for using the same macro/scrap names I have used in previous contexts. For example -- in Modula-3 it is customary to declare the publicly available attributes and methods of a particular module in the type Public, so there would be a Bingo.Public for module Bingo, a Blap.Public for module Blap, and so forth. Ada and Modula-3 (and probably other languages, as well) have no problem keeping these definitions separate because they are declared within a particular package or module which is logically an extension of their name.

I would like to be able to create an literate program containing several such encapsulations and keep these declarations separate within each context. However, in my admittedly limited survey of literate programming systems I have found none in which the concept of declaring such a context (as a hidden uniqueifier for macros, I suppose) is straightforward. If necessary, I suspect I could make the requisite changes to nuweb or FunnelWeb, but before I take on such a project I'd like to have a greater confidence that I'm not missing something obvious. Any suggestions?

From: **Lee Wittenberg**
Date: 29 Apr 1994

Mike Elliott writes: In working with languages such as Ada and Modula-3 which already have good encapsulation, I find a need for using the same macro/scrap names I have used in previous contexts. For example -- in Modula-3 it is customary to declare the publicly available attributes and methods of a particular module in the type Public, so there would be a Bingo.Public for module Bingo, a Blap.Public for module Blap, and so forth. Ada and Modula-3 (and probably other languages, as well) have no problem keeping these definitions separate because they are declared within a particular package or module which is logically an extension of their name.

I would like to be able to create an literate program containing several such encapsulations and keep these declarations separate within each context. However, in my admittedly limited survey of literate programming systems I have found none in which the concept of declaring such a context (as a hidden uniqueifier for macros, I suppose) is straightforward. If necessary, I suspect I could make the requisite changes to nuweb or FunnelWeb, but before I take on such a project I'd like to have a greater confidence that I'm not missing something obvious. Any suggestions?

If you're considering Nuweb or FunnelWeb, and the Ada/Modula-3 mechanisms do the job, why don't you just use these mechanisms in your program and let the compiler (rather than the literate programming system) do the work? Maybe I don't completely understand the question (a highly probable situation), but it seems to me that literate programming constructs should *supplement* traditional language features, not replace them. If you are trying to use literate programming to bring a Modula-3 style of programming to Pascal (for example), then I think you're going to be disappointed -- that's really outside of the literate programming ``problem domain'' (IMO).

I ran into the same thing recently when I tried to use literate programming to introduce Eiffel-style assertions and programming by contract to my C++ programs. I found no way to do this reliably and consistently, and I was (reluctantly) forced to the conclusion that, alas, there are some programming problems that literate programming cannot solve.

From: **Tommy McGuire**
Date: 29 Apr 1994

Mike Elliott writes: In working with languages such as Ada and Modula-3 which already have good encapsulation, I find a need for using the same macro/scrap names I have used in previous contexts.

How about this for an argument that reusing scrap names is not necessarily a good idea: You mention using similar identifiers in different modules because, due to the design of the language, the name spaces of the modules are separate. This works well because the languages support a fine, detailed structure. However, producing a literate program means producing a document (an article, report, book, or whatever) in a natural language. In all of the natural languages I am familiar with, the language does not support such a structure in a document. The language may allow a complex structure in a particular document but heavy structuring is not required nor particularly common. As a result, the name space for things like section titles, chapter titles, and scrap names is not smaller than the whole document.

If you reuse scrap names, you run the risk of confusing the reader as much as you would if you reused section names within chapters. The alternative to confusing him would seem to be using less meaningful, more generic scrap names similar to the ever popular "Introduction"-like sections, which I would prefer to avoid since the scraps are unique by definition and the ability to find a particular scrap is vital. All that being said, I must admit that in my own programs I have a large number of silly scraps like "Foo's variables," the proliferation of which might well be as confusing as anything else.

Macro preprocessing in language-independent WEBs

From: **Gregory Tucker-Kellogg**
Date: 30 Apr 1994

I'm just learning noweb and nuweb. As I understand it, the original WEB distinguished between what Knuth called "sections" and what he called "macros". Both constructs added readability to the woven output. Later, CWEB did away with macros, since the C preprocessor handled them anyway. I'm just learning noweb and nuweb and trying to decide bewtween them; both noweb's "chunks" and NUWEB's "macros" seem analogous to WEB's "sections". Has Knuth's original "macro" construct vanished for good? I'm not sure in what sense macro expansion is a language-dependent feature.

From: **Jeffrey McArthur**
Date: 30 Apr 1994

Gregory W. Tucker-Kellogg writes: Has Knuth's original "macro" construct vanished for good? I'm not sure in what sense macro expansion is a language-dependent feature.

Absolutely not. By choice, I am using WEB and Pascal. I won't say it is the original WEB designed by Knuth, because I have made some minor changes to it (unit, implementation, interface, and so on are reserved words and come out in bold when Woven and typeset; also some minor changes to Tangle). The original WEB had numerical macros. That is the tangle preprocessor would do math. This was to overcome the deficiency in Standard Pascal that you could not define a constant as a constant expression. For example, standard Pascal will not compile:

```
CONST
    TWO = 1 + 1;
```

Borland Pascal stole a lot from Modula-2 and now allows that. So numerical macros are going to pass from the scene. I don't think anyone is using them. Literate programming has profoundly changed the way I program. Now I get angry at myself if I run into a section of code I have not properly documented.

From: **Lee Wittenberg**
Date: 30 Apr 1994

Gregory W. Tucker-Kellog writes: I'm just learning noweb and nuweb. As I understand it, the original WEB distinguished between what Knuth called "sections" and what he called "macros". Both constructs added readability to the woven output. Later, CWEB did away

with macros, since the C preprocessor handled them anyway. I'm just learning noweb and nuweb and trying to decide bewtween them; both noweb's "chunks" and NUWEB's "macros" seem analogous to WEB's "sections". Has Knuth's original "macro" construct vanished for good? I'm not sure in what sense macro expansion is a language-dependent feature.

Yes, noweb's ``chunks'' are analgous to CWEB's ``sections.'' I don't know about Nuweb. Noweb is designed around the Unix idea of tools that work together. Macro processing is not part of noweb, but it can be provided by other, specialized tools. If you want to use C-type macros, you can pipe notangle's output through CPP or M4 before feeding it to your compiler. Of course, if you're using a language, like C, that supports macros directly, you don't need the extra step in the pipeline.

From: **Gregory Tucker-Kellogg**
Date: 30 Apr 1994

Lee Wittenberg kindly writes: Yes, noweb's ``chunks'' are analgous to CWEB's ``sections.'' I don't know about Nuweb. Noweb is designed around the Unix idea of tools that work together. Macro processing is not part of noweb, but it can be provided by other, specialized tools. If you want to use C-type macros, you can pipe notangle's output through CPP or M4 before feeding it to your compiler. Of course, if you're using a language, like C, that supports macros directly, you don't need the extra step in the pipeline.

One of the nicest features about TeX is its portability. Is a unix-dependent languge-independent WEB an oxymoron? I can certainly use m4, but I was wondering why macro expansion was not included in noweb and nuweb. At first blush, it could even be used as a sort of language-independent prettyprinting. Just italicize macro names in the woven output. The problem I'm working on is a collection of programs for controlling a spectrometer. The programs tend to be complex, but the final language is quite primitive, so a literate approach seem ideal. The language is so primitive, in fact, that macro expansion would give an enormous boost to code readability.

Is there a CWEB for IBM BookManager?

From: **Stephen Boyan**
Date: 05 May 1994

IBM has a product - Bookmanager - which offers both serial and hypertext, and uses Document Composition Facility (DCF) to typeset output for IBM 3800 laser printers. Is there a straightforward, or already existing, way to substitute DCF for Tex in CWEB? This would be usable across multiple IBM platforms, not just the mainframes.

From: **Lee Wittenberg**
Date: 05 May 1994

I don't know of any such thing already available, but it should be relatively straightforward to modify CWEAVE (there are a lot of places you'll need to make changes, but they should all be simple string substitution). You'll probably have to ``bell the cat'' yourself. You might also want to look into noweb, as well. It's fairly easy to write a new back end for any document processor you like.

Printing CWEB title to an odd-numbered page?

From: **Matt Pharr**
Date: 06 May 1994

I'll soon be printing a WEB to a schnazzy printer which prints on both sides of the paper. Since CWEB puts the title page last, though, I'd like to be sure that the title prints on an odd-numbered page, so that I can still put it in front of everything else and have things look good. Any suggestions for how to do this? All that I need to do, I think, is have some TeX like '\ifodd\pageno\vfill\eject\fi' spit out after the index but before the title, but I can't figure out how to do this--the CWEB syntax doesn't seem to offer any options, and a quick scan through the cweave web didn't bring out any simple ways to do this...

From: **Thomas Herter**
Date: 06 May 1994

I am doing such thing in following way: Generate some \TeX-output and find out the web-macros in it you wish to do something

``special inidividual'' for you, e.g. new page. Take webmac.tex resp. cwebmac.tex (dependent on the WEB you use) and find these macros in this webmac file. Do not modify the original files. Create your own e.g. ``mattweb.tex'', redefine the web macros in this file and simply include it in the preamble of your web-source. I am doing quite often such things e.g. to remove the mathematical sign for `mod' `and' `or' (I prefer bolded keywords) or to change the fonts or to create my own indenting convention. It works great and can be prepared with only a little effort.

From: **Lee Wittenberg**
Date: 06 May 1994

Matt Pharr asks: I'll soon be printing a WEB to a schnazzy printer which prints on both sides of the paper. Since CWEB puts the title page last, though, I'd like to be sure that the title prints on an odd-numbered page, so that I can still put it in front of everything else and have things look good. Any suggestions for how to do this?

If you're using CWEB 3.0 or 3.1, there is a commented-out macro that does just that. All you have to do is remove the `%' from the start of the line. If you're using an older version, why the heck havent you upgraded? ;-)

TeX is uncompromising in its quality

From: **Jeffrey McArthur**
Date: 08 May 1994

I strongly disagree with many of the arguments made by Eric W. van Ammers statements in the FAQ. The argument is made that literate programming tools should be language independant. That is fine, but show me any tools that deal with TeX itself, SGML, PAL, xBase, Lex, or Yacc? Knuth's original article says: "... Neither type of language can provide the BEST DOCUMENTATION by itself; but when both are appropriately combined, we obtain a system that is much more useful than either language separately."

TeX is uncompromising in its quality. One of the ideas behind TeX was to create a program that could typeset as well as some of the classic examples of typesetting. That is why ligatures, kerns, and the elaborate paragraph formatting were put into TeX. It is easy to compromise on that (look at any of the DTP systems and see the compromises).

I have tried noweb, fweb, FunnelWeb, and many of the other so-called literate programming tools. I have found them very lacking in the quality of the output. I want pretty printed output. I want the pretty printer built into the tools I use. I don't find that with those programs. Consider the case of SGML. A Document Tag Definition is not just a peice of text to include verbatim. There should be cross references. The index of a DTD is probably more important than the DTD itself. Show me any of the language independant literate programming tools that will deal with a DTD properly.

I created a quick little program to format TeX. It allows mixing of documentation and macros into a single file. I called this LitTeX. Although a few people beta tested it, I received almost no feedback (only one message even acknowledge they received the files).

The formatting, indexing, and cross-referencing needs of things as dissimilar as SGML and xBase make it very difficult to fit into a single program. Lex and Yacc are not supported at all. Let alone more obscure tools like CoCo. If you are programming is a language where a Lex/Yacc grammer are available. It is about a day's work to generate a pretty printer. Mixing that in with scrap management is a bit more work.

Writing portable code - any suggestions?

From: **Felix Gartner**
Date: 13 May 1994

I'm currently writing a program in C that should run on a variety of UNIX machines. Can anybody point me to a book or document that is especially concerned with writing _portable_ code. (I'm using CWEB, but that doesn't remove the more `technical' problems of porting programs, I believe 8-) Any help would be much appreciated,

---

From: **Andreas Stirnem**
Date: 14 May 1994

Can anybody point me to a book or document that is especially concerned with writing _portable_ code.

I find "Advanced UNIX programming" by Marc Rochkind excellent in this respect.

From: **Stephen Fulling**
Date: 15 May 1994

Can anybody point me to a book or document that is especially concerned with writing _portable_ code.

Portability and the C Language, by Rex Jaeschke, Hayden Books, 1989, ISBN 0-672-48428-5.

From: **Lee Wittenberg**
Date: 16 May 1994

Felix Gartner writes: I'm currently writing a program in C that should run on a variety of UNIX machines. Can anybody point me to a book or document that is especially concerned with writing _portable_ code. (I'm using CWEB, but that doesn't remove the more `technical' problems of porting programs, I believe 8-)

Unfortunately I'm away from my personal library at present, so I can't give you an in-depth answer (although I'm sure others will), but Kernighan & Plauger's "Elements of Programming Style" has some worthwhile stuff to say about portability, and "Guide to Good Programming Practice", edited by Brian Meek (I forget the publisher) also has some good guidelines, as I recall.

I would also like to mention a CWEB technique that I've had a great deal of success with respect to writing programs intended for more than one target. I use literate programming sectioning in connection with C #ifdef's to separate the non-portable parts from the portable stuff. For example, assume that I want to write a program that has to run on both Unix and MS-DOS. I'd structure it something like this:

```
%---- start of example
@*1The Important Stuff.
This is where the portable code goes. Somewhere in here I'll use a
system-dependent chunk.
...
@<This is a system-dependent chunk@>
...
@*1System-Dependent Code.
The system-dependent stuff goes here.
@*2Unix-dependent code.
This is where we put the Unix-dependent stuff (obviously). In this
simplified example, we only have one system dependent chunk, but in a
more ambitious program there would be many such chunks. Each chunk
in this ``chapter'' must be wrapped in a #if or #ifdef, like so:
@<This is a system-dependent chunk@>=
#ifdef UNIX
        /* Unix code goes here. */
#endif
@*2DOS-dependent code.
We do the same thing with the DOS code (explained separately,
naturally).
@<This is a system-dependent chunk@>=
#ifdef MSDOS
        /* DOS code goes here */
#endif
@ Often, one version needs variables, functions, etc. that the others
needs a global variable, |xxx|:
@<Global variables@>=
#ifdef MSDOS
        int xxx;
#endif
%---- end of example
```

That's the general idea. You can port to other systems simply by adding more subsections with apporpriate code. Actually, you can get a lot more sophisticated with this techinique, but these ``bare bones'' are enoughh to get you started. I've got a (half-finished) sample program that uses these techniques to the fullest. I'm going to try to get it in shape and make it available over the net some time this summer.

From: **Daniel Simmons**
Date: 16 May 1994

Felix Gartner writes: I'm currently writing a program in C that should run on a variety of UNIX machines. Can anybody point me to a book or document that is especially concerned with writing _portable_ code. (I'm using CWEB, but that doesn't remove the more `technical' problems of porting programs, I believe 8-)

I'm by no means an expert, but lately I have been looking into a few packages which help in this area. Two of the most popular ones are dist-3.0 and GNU's autoconf. I've been moving toward dist since it gives more control to the programmer in setting up his distribution. What I mean is that both scripts have the user run a configure script to localize the package to their site, but with dist you can better control that configure script and even have it ask the user specific questions, etc. while with autoconf it is pretty much an automated process that checks for the same set of systme features regardless of how you set it up. I would be interested in any other info you find on this topic. We might want to take this discussion to email or another group, though, since it doesn't directly deal with literate programming. (Shoot, some of these people even use--horror of horrors--DOS. :-)

From: **Stuart Ferguson**
Date: 16 May 1994

Felix Gartner writes: I'm currently writing a program in C that should run on a variety of UNIX machines. Can anybody point me to a book or document that is especially concerned with writing _portable_ code. (I'm using CWEB, but that doesn't remove the more `technical' problems of porting programs, I believe 8-)

While this doesn't answer your question, I wanted to mention that using literate programming does remove some of the problems with writing portable code. I'm currently writing some libraries which must function on widely different systems. In normal C the code would be filled with "#ifdef <system-type> ... #endif" blocks to implement the system-specific parts of the code. These normally clutter up the flow of the code making it hard to separate out the different paths that the code might take in different environments. Using a web I have been able to conceptually isolate these parts of the code. The first part of the document consists of system-generic code -- the stuff that is the same on all systems. This code is sprinkled with references to things like: < System-specific code to do something > I then have later chapters which implement the different systems. In the chapter on system-X, I would have:

```
<< System-specific code to do something >> =
#ifdef SYSTEM_X
        ... code with system-x calls ...
#endif
@
```

There would also be chapters on any other system that the code has been ported to. This organization has several advantages. It shows me the base functionality and structure without the complexities of different system calls. I can see a single system implementation in context, without any other systems cluttering things up. It also shows at a glance how complex the system-specific part of the code is compared to the system-generic part. The more that I can move to the system-generic part, the easier the code is to port. Finally, when I'm porting to a new system, I have a template immediately ready to show the parts I need to fill in for the new system. Just one of the latest niceties of web-structure that I've discovered, so I wanted to share it.

From: **John Evans**
Date: 17 May 1994

Felix Gartner writes: I'm currently writing a program in C that should run on a variety of UNIX machines. Can anybody point me to a book or document that is especially concerned with writing _portable_ code. (I'm using CWEB, but that doesn't remove the more `technical' problems of porting programs, I believe 8-)

Another extremely useful tool is "imake". It has been used with such large projects as X, Khoros, and Kerboros. I use it for my somewhat smaller tasks. But I wouldn't advise tackling it without the eminently readable "Software Portability With Imake," by Paul Dubois and available from O'Reilly and Associates. I use it with FWEB 1.30 and C.

From: **Tony Coates**
Date: 20 May 1994

Felix Gartner writes: I'm currently writing a program in C that should run on a variety of UNIX machines. Can anybody point me to a book or document that is especially concerned with writing _portable_ code. (I'm using CWEB, but that doesn't remove the more `technical' problems of porting programs, I believe 8-)

For getting software to be portable over UNIX systems, I find the GNU `autoconf' utility very good. The idea is that, for instance, if you need a particular function, you can always write a small test program and see if it links or not. If it does, you have it, if not, you don't. Then you can set -DHAVE_FUNC=1 for the compiler if you have the function, and the code is automatically adjusted to use the function. Basically, `autoconf' gives you a set of functions for doing such tests to configure a package for what is available on a system. To use it, you require a little knowledge of the `m4' macro package, unless you are lucky enough that the supplied functions can test all the things you need (there are quite a lot, so you may never need to worry about using m4, but you need to have it). Unfortunately, because `autoconf' writes a configuration test script designed for the Bourne shell, it is not so convenient for non-UNIX systems, though the same basic idea could be used for any system that supports some kind of batch language. Not that this is literate programming, but I need to make my literate programs portable as well ;-) .

From: **Felix Gartner**
Date: 22 May 1994

As some people have expressed their interest in the responses I got to my request, here's a short summary: I'm currently writing a program in C that should run on a variety of UNIX machines. Can anybody point me to a book or document that is especially concerned with writing _portable_ code. (I'm using CWEB, but that doesn't remove the more `technical' problems of porting programs, I believe 8-)

Recommended Books:

1) Rex Jaeschke: "Portability and the C Language." Hayden Books, 1989, ISBN 0672484285

2) Lapin, J. E.: "Portable C and UNIX system programming." Prentice-Hall, 1987. [Has a detailed overview over differences of tools and portability of systems calls between the different UNIX versions.]

3) Kernighan & Plauger: "Elements of Programming Style." 2nd ed, McGraw-Hill, 1978.

4) Brian Meek (ed.): "Guide to Good Programming Practice", ISBN 0470274174

5) Marc Rochkind: "Advanced Unix Programming", ISBN 0130118001

6) Paul Dubois: "Software Portability With Imake"

CWEB technique: Furthermore Stuart Ferguson and Lee Wittenberg both suggested a CWEB technique that I find very interesting. It deals with the problem of ugly #ifdef... #endif constructs in system dependent parts of the code. With the use of literate programming sectioning you can separate the non-portable parts from the portable ones (and even organize the the non-portable parts very clearly around the target architecture). Here's the example taken from Lee Wittenberg's posting (I've added a few blank lines for clarity):

```
-------- start of example -----------------------------------

...assume that I want to write a program that has to run on
both Unix and MS-DOS. I'd structure it something like this:

@*1The Important Stuff.
This is where the portable code goes. Somewhere in here I'll use a
system-dependent chunk.
...
@<This is a system-dependent chunk@>
...

@*1System-Dependent Code.
The system-dependent stuff goes here.

@*2Unix-dependent code.
This is where we put the Unix-dependent stuff (obviously). In this
simplified example, we only have one system dependent chunk, but in a
more ambitious program there would be many such chunks. Each chunk
in this ``chapter'' must be wrapped in a #if or #ifdef, like so:

@<This is a system-dependent chunk@>=
#ifdef UNIX
        /* Unix code goes here. */
#endif

@*2DOS-dependent code.
We do the same thing with the DOS code (explained separately,
naturally).

@<This is a system-dependent chunk@>=
#ifdef MSDOS
```

```
        /* DOS code goes here */
#endif

@ Often, one version needs variables, functions, etc. that the others
needs a global variable, |xxx|:

@<Global variables@>=
#ifdef MSDOS
        int xxx;
#endif

------- end of example ---------------------------------------------
```

Other things: Andrew Mauer points out that requiring something like ANSI C or GNU C will limit you. It seems that only K&R C won't cause problems. Daniel Simmons, Andrew Mauer and Tony Coates mention two software packages that can help to write portable code. One is dist-3.0 and the other is GNU's autoconf.

From: **Christian Steinmann**
Date: 03 Jun 1994

While this doesn't answer your question, I wanted to mention that using literate programming does remove some of the problems with writing portable code. ... Using a web I have been able to conceptually isolate these parts of the code. The first part of the document consists of system-generic code -- the stuff that is the same on all systems. ...

I don't need a web to isolate machine dependent parts in my code. I only have to choose a proper modular design to create code files which contain the pure ANSI code and few others which contain functions to cover the machine dependent features. Web and literate programming is a good idea, but I don't see a special profit in using web in a situation like that one, described above.

From: **Stuart Ferguson**
Date: 07 Jun 1994

I wrote: Using a web I have been able to conceptually isolate these parts of the code. The first part of the document consists of system-generic code -- the stuff that is the same on all systems. ...

Christian Steinmann writes: I don't need a web to isolate machine dependent parts in my code. I only have to choose a proper modular design to create code files which contain the pure ANSI code and few others which contain functions to cover the machine dependent features.

I'm writing a system-generic windows and graphics library. The entire project consists of the "few functions" which cover the machine-dependent features. The differences go way deeper than just syntax -- the very models are often incompatible. I have found literate programming to be a very powerful organizational tool in this case for the reasons I stated in my first message.

DVI specification and description

From: **Brian Edginton**
Date: 13 May 1994

Can anyone point me to a spec and description of DVI? I have tried archie, but all I seem to be able to get is *.dvi. A second question would be, is DVI the best device independent format currently available?

From: **bbeeton**
Date: 14 May 1994

*Can anyone point me to a spec and description of DVI? I have tried archie, but all I seem to be able to get is *.dvi.*

The description of .dvi can be found in the web source code to either tex (tex.web -- published in knuth's "tex: the program") or dvitype (dvitype.web). These files can be found on one of the ctan nodes (ctan = the comprehensive tex archive network). To find the file you want, after you've logged in as anonymous, ask for 'quote cite index name' where "name" can be any file name or name fragment.

From: **Mike Elliott**
Date: 15 May 1994

Brian Edginton writes: A second question would be, is DVI the best device independent format currently available?

Probably not. The format of a .dvi file was developed for TeX, and by and large was successful -- but a number of TeX weenies with whom I have corresponded, including some who were around during the early days, have said that if PostScript had been widely available at the time they were coming up with a device independent format they would have simply adopted that and not bothered to come up with their own. Anything you can do in a .dvi file you can do in PostScript, but the reverse is not even remotely true.

From: **Thomas Herter**
Date: 16 May 1994

The definition of the dvi-format can be found in dvitype.web. Weave this file and you will obtain the specification. The second question, if dvi is the best format for binary information exchange, can probably not be answered definitely without a careful comparision with other formats. The dvi format is intersting at least for following reasons (listed in random order):

a) It allows passage of integer numbers across different CPU architectures (low- high or med-endian).

b) It converts characters through its own conversion table, so that you can use a dvi file generated on a ASCII computer on (say) EBCDIC computer with no change.

c) A checksum is delivered as a consistency check.

d) A postamble contains pointers to pages, so that dvi-readers can quickly find a subrange of pages.

Since dvi format was defined in the eighties, there are probably a lot of better formats nowadays. Especially, some modern binary file formats for text of image storing support efficient data compression while dvi files tend to be large. I fear, you must judge about the best format by yourself.

From: **Lee Wittenberg**
Date: 16 May 1994

*Brian Edginton asks: Can anyone point me to a spec and description of DVI? I have tried archie, but all I seem to be able to get is *.dvi.*

"TeX: The Program" has a good description of DVI specs.

A second question would be, is DVI the best device independent format currently available?

"Best" is an adjective that starts religious wars. Best according to what criteria?

Who's using what?

From: **Lode Leroy**
Date: 17 May 1994

I wonder if there is any information on how many people are into literate programming, and using what tools. If it doen't exist, maybe someone could look into it. The people on linux-activists have made some kind of counter: every one sends his info to a special account, and some program calculates the statistics. Has anyone done this yet? If so, how can I get the information.

From: **John Scholes**
Date: 15 Jul 1994

Does anyone actually use literate programming for production code? In some ways I am a fan of literate programming: it makes for highly readable code. And presumably saves a fortune in the maintenance part of the cycle. But I have never managed to write one except by starting with an already finished program and converting it. [Maybe because I have never subscribed much to the top down school.] Even then, I have never persuaded anyone else to use it as the version they should maintain. How have the rest of you got on?

From: **Daniel Simmons**
Date: 16 Jul 1994

John Scholes writes: Does anyone actually use literate programming for production code?

Just starting to.

In some ways I am a fan of literate programming: it makes for highly readable code. And presumably saves a fortune in the maintenance part of the cycle. But I have never managed to write one except by starting with an already finished program and converting it. [Maybe because I have never subscribed much to the top down school.] Even then, I have never persuaded anyone else to use it as the version they should maintain. How have the rest of you got on?

I have recently adopted literate programming as standard practice. So far I have converted one system that was already into production and written two other systems from scratch literate (one in production for a while, another just finishing up and getting ready to go into production).

One of the biggest benefits I have found to using literate programming (aside from the fact that I write much better programs when I have to think about and justify what I'm doing) stems from the fact that I don't have the luxury of working on development all day every day. My time is more often than not directed to system administration, user support and twelve other tasks that distract me from the development projects I'm trying to complete. When I'm working on a literate program, I am able to begin making progress again much more quickly after a distraction than if I am working on a non-literate program.

This is not terribly surprising since it is essentially the same argument that is used for saying that literate programs are easier to maintain, but I hadn't thought of it until I found out by experience. Now if I could just convince more people around me to jump on the band- wagon...

From: **David Kastrup**
Date: 17 Jul 1994

John Scholes writes: Does anyone actually use literate programming for production code? In some ways I am a fan of literate programming: it makes for highly readable code. And presumably saves a fortune in the maintenance part of the cycle. But I have never managed to write one except by starting with an already finished program and converting it. [Maybe because I have never subscribed much to the top down school.] Even then, I have never persuaded anyone else to use it as the version they should maintain. How have the rest of you got on?

I have done one literate project from scratch in CWEB, but it required too much discipline to get everything typeset properly for a project which is not thought for publishing, but for development and documentation.

I have done several things in noweb, and it's more or less just it, as I can hack thing and documentation down without thinking too much, or worrying about looks. Noweb actually shortens my development phase, as the docs do not require additional thinking, and neither does the code layout. However, it took some work persuading noweb to let german.sty work properly. The TeX macro support could be more versatile. For English language programmers, this should not make much difference.

From: **Lee Wittenberg**
Date: 17 Jul 1994

John Scholes asks: Does anyone actually use literate programming for production code?

Yes. I do. I spent the 1992-93 academic year on sabbatical with a small software house, and wrote all my code using literate programming tools. These include programs written in C, Paradox, Paradox for Windows, and Awk. One of these (a PfW form "disassember" called d'Art) might even be the first non-Borland commercial application written for Paradox for Windows. I am back in academe, but I still write programs (some of which are for public consumption), and I still use literate programming (mostly noweb).

Carl Gregory, who is not a subscriber to this list, also uses literate programming for production code. Unlike me, Carl works for a commercial software firm. Because of Carl's positive experiences with literate programming (better, more maintainable code faster) the whole firm is moving (slowly) to becoming a completely literate programming house (delayed only by the usual reluctance of programmers to embrace anything that increases work "up front," even if it saves more time later in debugging and maintenance).

From: **Chris Nadovich**

Date: 18 Jul 1994

John Scholes writes: Does anyone actually use literate programming for production code?

In the recent past I have been the systems engineer overseeing two major Radar system developments that used literate programming. Both systems had considerable software content. In one case the software was mostly embedded, while in the other case there was a significant human interface. The size of the efforts were moderate as these things go (say 50,000-100,000 lines of C code each). Since I was in charge, and I did a lot of the work myself, I was able to force the use of literate programming for both efforts. In other programs that I have been involved in, where I didn't have the political strength I had for these two efforts, I _suggested_ the use of literate programming, but my suggestions were not followed.

The two radars that used literate programming had software delivered on-time and in-budget. Weave also generated comprehensive, low-level documentation that was invaluable when it came to supporting the systems in the field, not to mention giving us a few data-items for free. I should say that these developments were hardly "top-down", although the eventual literate programming design _was_ organized that way. There was considerable "middle-out" development. literate programming handled this just fine.

My major frustrations with literate programming (as posted here earlier) have had to do with the lack of scoping and modularity. In moderate to large efforts where a large team of programmers with mixed abilities must be coordinated it is convenient to encapsulate individual efforts in functions, classes, or files. The scraps that literate programming provides aren't robust enough, in my experience, for reliable re-use---yet inexperienced people that use literate programming tend to write a lot fewer functions, classes, or files, and break things up with scraps. This may result in pretty, readable code, but the scraps tend to have poorly defined interfaces and often rely on global variables for essential communication with distant scraps.

Another problem with literate programming in a production environment is its status as an "experimental" system. There's always an new variant posted and this is good, but I have to use just one version and stick with it for better or for worse over a 3-5 year development cycle. A corrilary of this problem is that the "real world" to a large extent does not know about literate programming. That means I have problems with language support and with training. For example, when I buy a DSP board for my radar and it comes with a C compiler, I generaly only have about a %50 chance of discovering that the #line directive is supported. Debuggers are often quite bad at supporting this language feature---Borland's Turbo Debugger is a classic example of missing #line support.

And finally, the training issue is probably the worst. It's easy for a PhD graduate student of CS to decide to use literate programming in his thesis, but consider the typical production programmer. They've spent the last five years hacking COBOL at some bank's sweat shop. They took a C course at a community college and finally landed a job to use C at a defense contractor. The first day on the job, this bozo called a "systems engineer" drops "The TeXbook", "K&R", "Strunk and White" and Knuth's "Literate Programming" on their desk and expects them to be experts in all three by Friday.

From: **Eric van Ammers**
Date: 18 Jul 1994

We certainly do use literate programming for production code, be it that we are a university and not a commercial organization. The grading administration of our department, originally written for DECsystem-10 (Pascal) and later rewritten for VAX/VMS (again Pascal) are literate programs. We have done a lot of maintenance on these programs and then one really appreciates this technique

---

From: **Bart Childs**
Date: 19 Jul 1994

I use it extensively. I have had lots of students use it too and generally there is no problem as long as they take a deep breath and start. We used it in a freshman class using: DOS, demacs (and now would do oemacs), and Turbo Pascal. Obviously we were using the original Pascal-WEB. Other than a few sour grapes people who wanted `pure Turbo' there was no problem.

I am convinced that even beginners can develop codes quicker and better if they will make discrete steps of: 1) Write a statement of the program and get it reviewed! 2) Of course that was done in WEB, now add the top level view of the program, like in pseudo code. Review it! 3) Do the whole thing! So you are by yourself, be sure to stop and print it in each of the steps and review it yourself. Just try to consciously think that you are not the reviewer not just the hacker.

TeX is not an editor. It is a formatter and the amount that is written by most people doing literate programming is nearly trivial.

Bad style in Stanford GraphBase

From: **Tommy McGuire**
Date: 27 May 1994

(What the heck, things have been kind of boring around here lately.) I have been reading Knuth's _The Stanford GraphBase_ in my free time lately (and trying not to dribble on it; the only free time I've had has been around dinner), and I've noticed a couple of problems with it. In particular, it seems like Knuth does some things that are very bad style, such as using or even defining some item and not describing until later (if ever) what it is or is used for. The worst example I have run across is in GB_GRAPH, section 31:

```
[...]
[blah, blah, stuff about gb_virgin_arc, gb_new_arc, and Arcs, blah]
#define gb_new_graph gb_nugraph /* abbreviations for Procrustian
                                   linkers */
#define gb_new_arc gb_nuarc
#define gb_new_edge gb_nuedge
[...]
```

The only time gb_nugraph and co. show up in the index is there and in section 41 of GB_GRAPH, where the #defines are set up for gb_graph.h. Any idea what these things do or are actually for? I can see how the shortened forms of the names would be good for broken linkers that only look at the first 6 characters of an identifier, but then wouldn't you need to define the actual functions as "gb_nugraph" rather than "gb_new_graph"?

I had another example, which was a little less bad, of Knuth using a function or data type without any description until much later, but I can't seem to find it now. To me, this seems to be one of the worst examples of bad style in a literate program. As a reader, I don't want to be kept in suspense unless there is a pressing reason like, for example, the explanation of the thing in question would be a significant digression from the current topic. Even then, in my own work, I generally find that there is a way of restructuring the program that avoids the problem.

From: **Jacob Nielsen**
Date: 30 May 1994

```
      [...]
#define gb_new_graph gb_nugraph /* abbreviations for Procrustian
                                   linkers */
#define gb_new_arc gb_nuarc
#define gb_new_edge gb_nuedge
[...]
```

The only time gb_nugraph and co. show up in the index is there and in section 41 of GB_GRAPH, where the #defines are set up for gb_graph.h. Any idea what these things do or are actually for? I can see how the shortened forms of the names would be good for broken linkers that only look at the first 6 characters of an identifier, but then

I guess you have answered your question (I'm not a wizard with broken compilers :-) If the defines are for the benefit of broken compilers, I would say that the comment in the code is sufficient! I wouldn't say that the use of comments in the code is the best way to document a literate program though. IMHO, things to please compilers should be placed in a seperate section or crossreferenced in a sensible way (using words, not references to function names!) (if possible; since DEK's CWEB can't use LaTeX, this could become a problem) Since gb_nugraph and co. are not part of the program proper but only there to please some compilers, there is no need to index them like variables and function names.

wouldn't you need to define the actual functions as "gb_nugraph" rather than "gb_new_graph"?

For DEK's sake, I hope not :-)

I had another example, which was a little less bad, of Knuth using a function or data type without any description until much later, but I can't seem to find it now. To me, this seems to one of the worst examples of bad style in a literate program. As a reader, I don't want to be kept in suspense unless there is a pressing reason like, for example, the explanation of the thing in question would be a significant digression from the current topic. Even then, in my own work, I generally find that there is a way of restructuring the program that avoids the problem.

Unless the meaning of the function/data type is clear from the context, I agree that it is bad style. Perhaps I should read Stanford GraphBase; at least to see if I agree with Tommy :-)

From: **Gregory Tucker-Kellogg**
Date: 30 May 1994

Tommy Marcus McGuire writes: (What the heck, things have been kind of boring around here lately.) I have been reading Knuth's

The Stanford GraphBase in my free time lately (and trying not to dribble on it; the only free time I've had has been around dinner), and I've noticed a couple of problems with it. In particular, it seems like Knuth does some things that are very bad style, such as using or even defining some item and not describing until later (if ever) what it is or is used for. [details deleted]

Doesn't DEK challenge readers to find errors in _The Standard GraphBase_, just as in his other books and programs? You might have just won $2.56!

From: **Dietrich Kappe**
Date: 30 May 1994

```
      [...]
  #define gb_new_graph gb_nugraph /* abbreviations for Procrustian
                                     linkers */
  #define gb_new_arc gb_nuarc
  #define gb_new_edge gb_nuedge
  [...]
```

The only time gb_nugraph and co. show up in the index is there and in section 41 of GB_GRAPH, where the #defines are set up for gb_graph.h. Any idea what these things do or are actually for? I can see how the shortened forms of the names would be good for broken linkers that only look at the first 6 characters of an identifier, but then

If the #define is in effect when "gb_new_graph" is *declared*, then what is actually being declared is "gb_nugraph" (think textual replacement).

From: **Tommy McGuire**
Date: 31 May 1994

*Dietrich Kappe writes: If the #define is in effect when "gb_new_graph" is *declared*, then what is actually being declared is "gb_nugraph" (think textual replacement).*

Well, I'll be a.... I should have thought of that. Maybe there is a down side to the ability to rearrange things in a program---I might have caught that one if the #define had been at the beginning of the file. On the other hand, maybe not.My original point remains: Knuth just dropped them in without bothering to explain them with more than a rather trivial comment.

From: **Ozan Yigit**
Date: 31 May 1994

Tommy McGuire writes: I have been reading Knuth's _The Stanford GraphBase_ [...] [...] and I've noticed a couple of problems with it.

You are so gentle. The book is a bucket full of algorithmic diamonds and pearls buried in a disaster area littered [or was it "literate"] with poor typesetting and programming. Required inscription: lasciate ogni speranza, voi ch'entrate. oz (hurriedly slips into a flame-retardant suit...)

From: **Marc van Leeuwen**
Date: 01 Jun 1994

Tommy McGuire writes: I have been reading Knuth's _The Stanford GraphBase_ [...] it seems like Knuth does some things that are very bad style, such as using or even defining some item and not describing until later (if ever) what it is or is used for. The worst example I have run across is in GB_GRAPH, section 31:

```
  [...]
  [blah, blah, stuff about gb_virgin_arc, gb_new_arc, and Arcs, blah]
  #define gb_new_graph gb_nugraph /* abbreviations for Procrustian
                                     linkers */
  #define gb_new_arc gb_nuarc
  #define gb_new_edge gb_nuedge
  [...]
```

The only time gb_nugraph and co. show up in the index is there and in section 41 of GB_GRAPH, where the #defines are set up for

gb_graph.h. Any idea what these things do or are actually for? I can see how the shortened forms of the names would be good for broken linkers that only look at the first 6 characters of an identifier

and in a later message adds

On the other hand, my original point remains: Knuth just dropped them in without bothering to explain them with more than a rather trivial comment.

I had no problem with this comment; if anything it is a bit cryptic rather than trivial. Your initial analysis is quite correct, and these lines are only there to accommodate linkers that only inspect a few initial characters of identifiers (such linkers are a nuisance, but not broken according to the ANSI/ISO standard, as long as they inspect at least 6 characters, possibly ignoring their case). Since this is just a minor portability issue (without these lines the programs will run just as well on a system whose linker does a decent job) it does not seem to require a lot of explicit attention. On the other hand, if you are aware of Knuth's style, you will know that such a comment may contain some subtle humour, as it does in this case. For those who missed the point, the reference is to the mythological chap named Procrustus who had the unpleasant habit of providing his guests with a bed whose size was ill fit to that of the guest, and then to adapt the size of the guest to that of the bed. Once the reason for these #define-s is clear, it is not difficult to see that one may immediately forget about them. There may be one exception to this though, in case you wish to use a symbolic debugger on this program: these are usually unaware of the source text before preprocessing (although they may show its lines), so to them one must refer to things like "gb_nuarc" instead of "gb_new_arc".

From: **Thomas.Herter**
Date: 07 Jun 1994

Ozan Yyigit writes: You are so gentle. The book is a bucket full of algorithmic diamonds and pearls buried in a disaster area littered [or was it "literate"] with poor typesetting and programming.

Since the beginnings of Web Knuth has been criticized again and again, but mostly only on the paper. I remember in this context J. Bentley's articles in CACM with an introduction to Web and a criticism by Doug McIllroy (which was more a clumsy reclame for Unix than a disscusion about Web). Nobody has delivered a better tool for literating programming. Although criticism is very welcome. Meanwhile after ten years of practice with the Web-family of tools it should be based on... new and better programs. I believe, that criticism of this kind, like the above is not fair. To disscuss, that some constants were not used in the text or just to claim that the book if full of ``poor programming'' or similar things is very superficial. Feel free to publish algorithmic diamonds typesetted after your taste. As an example of such alternative effort to literate programming take Holub's book "Compiler Design in C". Holub uses two self designed tools `autopic' and `arachne' to typeset a whole book inclusive citations of the c-sources and picture inclusion for troff and pic. While Knuth prefers a "math-like" look of his chunks of programs, Holub is more "listing-oriented". The programs in his book look like regular listings, although split into the order corresponding to the order of the verbal explanations.

What I try to point out, is the fact, that literating programming is partly an individual process. Knuth said in foreword to many Web-examples, that he is not claiming to have discovered a kind of silver bullet to literate programming. Web is just his initial idea to document programs or to write books describing guaranteed correct and running code. Everybody if free to improve his tools or even to find out a better way to write literate programs. For example, I also do not like the optical appearance of the parts of code in original Web. Instead of criticizing Knuth, I wrote a modified webmac.tex and use my own macros instead. I am accepting the fact that Knuth has a different taste and that he has a damn good right to have individual taste because he is a creator of the tools given as a gift for all of us and he is a creator of the term `literate programming'.

No other author of literature and programs has combined the theoretical skills and implementation effort like Donald Knuth (N. Wirth should also be named in this context). His code is free and of uncomparable portability and quality. Of course, nobody's code is free of mistakes. Almost every bigger sized program can be improved or redesigned. Knuth is very open to concrete proposals of improvements, he is collecting lists of mistakes and is trying to analyse the reasons for the mistakes---look at the article "The Errors of \TeX", an unique effort to analyse the whole period of \TeX\ developement including diagrams showing the number of bugs found and corrected in every stage of the developement. If you have found bugs in the Stanford Graphbase or if you believe to have found poor algorithms, just be precise and explain what kind of improvements you would like to propose.

Multiple module references

From: **Robert Partington**
Date: 09 Jun 1994

Not that I want to keep posting this question, but... Is it possible in any literate programming scheme to have multiple module

references in one section? (preferably CWEB) I mean...

```
@ this section opens the file for reading

@<|main| local variables@>=

FILE *fhandle;

@<Open the fil...@>=

fhandle=fopen("file","r");

etc.
```

I don't like having unnamed sections tied to the previous one (purely for stylistic reasons), and I think that declaring local variables where they are used is more in keeping with the literate programming philosophy. I did try and hack cweave to do this (by adding @9 and making the section translator loop while it got @9's - .ch file available on request) but the section numbers of any defining references comes out too high. (ie in the above example, the @<Open...@>= section number is 1 too high). Any help, ideas, money :) gratefully received.

From: **Andreas Stirnem**
Date: 10 Jun 1994

Is it possible in any literate programming scheme to have multiple module references in one section? (preferably CWEB)

Noweb allows this syntax. For instance:

```
----------------- begin example -------------------------
@ Access homogeneous parts.
<<preliminary class declarations>>=
class hpol;

<<local user header files>>=
#include "hpol.h"               // from hpol import hpol

<<public members>>=
hpol operator()(const int d) const;

<<*>>=
hpol series::operator()(const int d) const
{
        /* ... */
}
------------------ end example ----------------------------
```

For me, this was a reason for switching to noweb. I do think that this is an important feature. I always felt that the refinement structure is only a part of the story. There is complementary structure you might call the ``boiler plate'' with chunk names like.

```
<<system header files>>
<<user header files>>
<<class declarations>>
<<function prototypes>>
<<inline functions>>
```

You create these standard slots and afterwards emit code into them. I think it is good style to keep semantically related code close together in the source file, even if it is going to be spread out over the whole compilable file.

From: **Jacob Nielsen**
Date: 10 Jun 1994

Robert Partington writes: Not that I want to keep posting this question, but... Is it possible in any literate programming scheme to have multiple module references in one section? (preferably CWEB)

Yes; at least it works in nuweb and noweb -- the two tools I have tried. This is because these tools does not enforce a particular style

that the document should have. CWEB (and WEB) enforce a style which, if I understand you correctly, only allows for one module reference per section. Perhaps the LaTeX support for CWEB (by Joachim Schrod, I believe) eliminates the problem. There is also another CWEB called cweb-leeuwen -- I cannot remember what it can do but you could get lucky :-)

[example deleted] I don't like having unnamed sections tied to the previous one (purely for stylistic reasons), and I think that declaring local variables where they are used is more in keeping with the literate programming philosophy.

Hear, hear!

I did try and hack cweave to do this (by adding @9 and making the section translator loop while it got @9's - .ch file available on request) but the section numbers of any defining references comes out too high. (ie in the above example, the @<Open...@>= section number is 1 too high).

When you say the <Open...> section number is 1 to high, does that mean that you want the section numbers of <|main|...> and <Open...> to be the same ? If that is the case, then what happens with the crossreferences?

From: **Eric van Ammers**
Date: 16 Jun 1994

I'm not sure if I understand your problem correctly, but if so then CLiP solves your problem. In the CLiP system there is predefined connection between the structure of your the documentation (Chapter, sections, subsections, etc) and the scraps of your literate program. Any compination of scraps inside sections is allowed. In a Pascal environment this allows you to simulate local declarations. That is if you have a refinement step with local variables you can document both together in one section while the declarations will still be inserted at their syntactically correct posistion at main level.

C to LaTex converter

From: **Graham Trigge**
Date: 09 Jun 1994

I am looking for a programming / pretty printer which will convert C code into LaTex code. I know such things are around for the SR programming language. If anyone knows of such a program, could they mail me at the below address.

From: **Lewis Perin**
Date: 09 Jun 1994

I think you want c2cweb *plus* CWEB (sorry, you need both.) Check the comp.programming.literate FAQ for ftp sites.

From: **Nelson Beebe**
Date: 09 Jun 1994

One solution is tgrind, an updated version of which is available on ftp.math.utah.edu in /pub/tex/pub/tgrind; tgrind knows about several languages, and produces TeX output that can be easily incorporated in LaTeX documents.

Another solution is quick and dirty: make a 2-line header that looks like this:

```
@
@c
```

Copy this into an empty file, append a C source file to it, and run it through cweave, e.g.

```
echo "@" >foo.w
echo "@c" >>foo.w
cat myfile.c >>foo.w
cweave foo
tex foo
```

I expect that very similar tricks can be used with other literate programming tools that support C or C++.

Reverse-engineering code

From: **Tony Coates**
Date: 16 Jun 1994

I was just reading an article in Unixworld's Open Computing about tools for reverse-engineering software. The crux of it was that these tools create a database which describes the structure of the code and the dependence of global data structures, etc. Many then provide a base from which to continue the development of the software, with the tool essentially tracking the program structure as the development process continues, sometimes even enforcing coding conventions before allowing a new revision of a piece of code to be registered.

It occurred to me that this style of tool could well be the way in which literate programming might end up being introduced to the `great unwashed'. In some ways, of course, it goes against the literate programming paradigm that code should be well documented from the start, but then not everyone has the advantage of starting from scratch. Then there are examples of LitProg tools being used for reverse engineering, such as when Ross Williams used FunnelWeb to add documentation to a Postscript program until he understood the structure of the program enough to allow him to fix the code. While this could be done with any appropriate LitProg tool, it's also a good bet that people would strongly prefer automatic code analysation to the manual process that Ross went through.

Tools like FunnelWeb, noweb, and nuweb, being language-independent, i.e. language insensitive, would not be the best place to look for such features to be added, but what about with CWEB or FWEB? Are there any such tools currently around? (I only use FunnelWeb, so I wouldn't know) Clearly designing such a tool is a major job, because you have to parse the code, but since CWEB, FWEB, and a few others already parse the code, would adding structure analysis of some sort be unreasonable? Not that I'm volunteering ;-) by the way, I'm just interested into how the current crop of LitProg tools will fit into some of the developing trends in code management. Any thoughts?

From: **Stephen Boyan**
Date: 16 Jun 1994

I was just reading an article in Unixworld's Open Computing about tools for reverse-engineering software. The crux of it was that these tools create a database which describes the structure of the code and the dependence of global data structures, etc.

It occurred to me that this style of tool could well be the way in which literate programming might end up being introduced to the `great unwashed'. In some ways, of course, it goes against the literate programming paradigm that code should be well documented from the start, but then not everyone has the advantage of starting from scratch. Then there are examples of LitProg tools being used for reverse engineering, such as when Ross Williams used FunnelWeb to add documentation to a Postscript program until he understood the structure of the program enough to allow him to fix the code. While this could be done with any appropriate LitProg tool, it's also a good bet that people would strongly prefer automatic code analysation to the manual process that Ross went through.

I do have some thoughts on this. In fact the use of C-Web with my favorite third party tools like PC-Lint and C-Vision is a question I've been holding in the back of my mind until I start trying to code in a literate style. (Right now I'm learning TeX, and still coding in normal C.) I have some experience with reengineering tools for COBOL on the IBM mainframe. I think they are assists to the maintenance programmer, but do not fully automate the job. The art comes in knowing what can be "assisted", and what must be done manually. I think reengineering code lies with tasks like indexing books; there is no fully automated way to do it. If you try to automate indexing, you get a concordance, not an index. Automated reengineering tools are also great at generating concordance-size output, instead of a more focused and distilled index-size.

BTW, I do not include tools like Lint with the COBOL tools of the previous paragraph, although Lint can also give copious output. I have found PC-Lint far more useful than any COBOL ree tool I've used to date. I also like breakpoint debuggers. So my thoughts regarding C-Web and third party automated tools can be summed up: how can lints and debuggers be used with literate programs? and how can code databases be analyzed and tied back to the original literate source? It would seem all these thrid-party tools would work on the C output from the original literate text, rather than the text itself, and that is the difficulty. The approach of manually adding "webbing" to the code until it becomes understandable might be assisted by third-party analyzers. I had a gut reaction to the mention of this above as a promising avenue of research and practice, and hope to follow it. Thank you.

CWEB formats C++ badly

From: **Anssi Porttikivi**
Date: 19 Jun 1994

Tell me, is CWEB 3.1 which we run here really supposed to parse non C but C++ constructs properly? I have a project which is in two dozen .w files and produces various .H and .C files. But e.g. class header files are never formatted right, we will have to include A LOT of CWEB formating commands to make line breaks sane. Indentations are impossible to fix properly. Templates confuse CWEB badly. Sometimes CWEB produces faulty C++ code (Try a<b<c> >, you will have to write it a<b<c>@= @>>). I am so tired with it...

From: **David Kastrup**
Date: 20 Jun 1994

Anssi Porttikivi writes: Tell me, is CWEB 3.1 which we run here really supposed to parse non C but C++ constructs properly? I have a project which is in two dozen .w files and produces various .H and .C files. But e.g. class header files are never formatted right, we will have to include A LOT of CWEB formating commands to make line breaks sane. Indentations are impossible to fix properly. Templates confuse CWEB badly. Sometimes CWEB produces faulty C++ code (Try a<b<c> >, you will have to write it a<b<c>@= @>>). I am so tired with it...

I recommend switching to a non-pretty-printing Web, such as noweb or nuweb. In my opinion (I have used both CWEB and noweb) the only excuse for using prettyprinting web's is for publication of a program (which has a very special focus on appearance, like typesetting a typesetter source :)), or for fast typesetting of machine-generated code using WEB constructs (if your generated code does not, you are better off with a special purpose prettyprinter like indent), or for writing code where the programmer has as bad a style as to be less readable than machine-formatted code.

In all other cases, the extra hassle is not worth it. This is especially the case with in-house documented source code. When the choice to the WEB programmer is "1) have lots of additional pains" or "2) do it illiterate" the products will look nice, for a while, then he'll drop the webs as the novelty wears off.

For puzzling together docs and programs, noweb for me does the trick very nicely, and I do not consider it more work doing it literate. Especially for syntactically complicated languages (like C++) with a host of slightly different compilers around, a pretty-printing Web is almost bound to fail eventually, as it does not have the compiler's complex parser, and not even the contexts for parsing. Instead of trying to coax a tool into mangling the code not too badly, better use one that does not even try, and hands the resposibility over to you, the human. Good pragrammers indent anyway, without thinking much.

From: **Lee Wittenberg**
Date: 20 Jun 1994

Anssi.Porttikivi asks: Tell me, is CWEB 3.1 which we run here really supposed to parse non C but C++ constructs properly? I have a project which is in two dozen .w files and produces various .H and .C files. But e.g. class header files are never formatted right, we will have to include A LOT of CWEB formating commands to make line breaks sane. Indentations are impossible to fix properly. Templates confuse CWEB badly. Sometimes CWEB produces faulty C++ code (Try a<b<c> >, you will have to write it a<b<c>@= @>>). I am so tired with it...

C++ is an extremely difficult language to parse, and certain constructs need to be dealt with differently than in C. The CWEB parser tries to be as simple as possible, so it misses a lot of ``strange'' constructs. I suspect that you are not making good use of the @[ and @] formatting commands. These ``brackets'' turn whatever is inside them into an expression for the purposes of formatting. Almost every C++ formatting problem I've had in CWEB was solved with a simple application of @[and @].

Can one have it all?

From: **Allan Adler**
Date: 20 Jun 1994

I have been looking over the various web programs available. It seems to me that there is no program that incorporates all of the best features of the availble web programs. Correct me if I'm wrong, but this is how it looks to me: FWEB can handle several languages well but it is not convenient for the user to add support for another language (although there is support for using any language

verbatim). Also, it can't write to multiple output files. Spidery web can handle new languages but not multiple output files. Noweb can handle multiple output files but not languages such as fortran where the position of tokens on a line is important.

I received a large fortran program that I would like to consider rewriting with a suitable web. THe program is spread out over several dozen files. It also involves some files written in C. What web does one use for something like that? I realize one can make a web file for each fortran file but then one loses information about the relation among the files.

From: **Norman Ramsey**
Date: 20 Jun 1994

Allan Adler writes: Spidery web can handle new languages but not multiple output files.

I've been writing lousy documentation again. Spidery web handles multiple output files. In fact, I invented the idea of multiple output files when I designed spidery web.

Noweb can handle multiple output files but not languages such as fortran where the position of tokens on a line is important.

Ditto. Noweb does indeed handle languages in which the position of tokens on a line is important (including not just fortran but miranda, haskell, python, and bourne shell here documents). And again it was the first literate-programming tool to do so. You'd be doing me a service if you could point me to whatever you read that gave you these misconceptions, so I could correct it.

I received a large fortran program that I would like to consider rewriting with a suitable web. THe program is spread out over several dozen files. It also involves some files written in C. What web does one use for something like that?

Either noweb or nuweb would be most suitable. If you want prettyprinting and are willing to tolerate a more complex tool, FWEB might also be good. (I would be surprised if it couldn't handle multiple output files and multiple languages simultaneously, but where FWEB is concerned I don't know what I'm talking about.)

From: **Gregory Tucker-Kellogg**
Date: 20 Jun 1994

Norman Ramsey writes: Either noweb or nuweb would be most suitable. If you want prettyprinting and are willing to tolerate a more complex tool, FWEB might also be good. (I would be surprised if it couldn't handle multiple output files and multiple languages simultaneously, but where FWEB is concerned I don't know what I'm talking about.)

FWEB does handle multiple output files. Look at the @o and @O commands for output files with local and global scope, respectively. FWEB can also compare temporary files (following the example of noweb) to avoid triggering an unnecessary make, but you should turn off the commenting options because the '#line nnn "foo.web"' insertions are likely to change even if you modify just the documentation part.

From: **Lee Wittenberg**
Date: 20 Jun 1994

Allan Adler writes: I have been looking over the various web programs available. It seems to me that there is no program that incorporates all of the best features of the availble web programs. Correct me if I'm wrong, but this is how it looks to me: FWEB can handle several languages well but it is not convenient for the user to add support for another language (although there is support for using any language verbatim). Also, it can't write to multiple output files. Spidery web can handle new languages but not multiple output files.

Spidery WEB *can* handle multiple output files, just not multiple languages in a single web.

Noweb can handle multiple output files but not languages such as fortran where the position of tokens on a line is important.

While this is basically true, the output from notangle can be piped through a postprocessing filter that can deal with line positions (an extremely simple sed script can deal with Fortran; other less simple filters can deal with Cobol and even assembly language!).

I received a large fortran program that I would like to consider rewriting with a suitable web. THe program is spread out over several

dozen files. It also involves some files written in C. What web does one use for something like that? I realize one can make a web file for each fortran file but then one loses information about the relation among the files.

I'd recommend noweb, but any of the language-independents should do the job. Also, I'm a bit surprised to hear you say that FWEB can't support multiple output files; I was sure it could (but, then again, I don't use FWEB, and nonusers are always the last to know :-).

From: **Sven Utcke**
Date: 21 Jun 1994

Allan Adler writes: Correct me if I'm wrong, but this is how it looks to me: FWEB can handle several languages well but it is not convenient for the user to add support for another language (although there is support for using any language verbatim). Also, it can't write to multiple output files.

Well, it used to be able to in version 1.30a (which I'm using). You can even have several different languages (e.g. c and a makefile) in one .web-file and have them write to several different files like .c, .h and the makefile (I do it all the time).

From: **Thorsten Ohl**
Date: 21 Jun 1994

Allan Adler writes: Noweb can handle multiple output files but not languages such as fortran where the position of tokens on a line is important.

That can be easily worked around. I just strip all leading whitespace from chunk references and use standard indentation in the chunk definition. Here's a snippet from one of my Makefiles:

```
NOTANGLE = sed 's/^[    ]*<</<</' | notangle

hktest.f: hepawk.nw
        cat hepawk.nw | $(NOTANGLE) -R'Test program' | cpif $@
```

From: **Mark Naumann**
Date: 21 Jun 1994

Allan Adler writes: Correct me if I'm wrong, but this is how it looks to me: FWEB can handle several languages well but it is not convenient for the user to add support for another language (although there is support for using any language verbatim). Also, it can't write to multiple output files.

I believe that FWEB's @O changes the global default (output) filename and @o changes the (output) filename for the current section/module. I'm relying on this in my code, so that I can put definitions in my header (.h) and code in the .c file. The speed of the leader determines the rate of the pack.

From: **Allan Adler**
Date: 22 Jun 1994

Thanks to all who replied to my posting. After reading the replies, I started wanting more features (e.g. I am under the impression that there are problems with webbing ANSI C's function prototypes). Rather than nickel-and-dime everyone to death, here is a more comprehensive question: is there (perhaps in a FAQ sheet) a table (i.e. a matrix) showing the different webs on the one hand and the different properties a web might be expected to have and showing which ones have those properties. To greatly oversimplify:

```
         WEB  CWEB  FWEB  noweb  nuweb FunnelWeb spideryweb RWEB schemeweb ...

mult.
output
files?

C

C++
```

```
Fortran

Pascal

Retar-
getable?

ANSI C?

Multiple
languages
simul-
taneously?

etc.
```

Is there such a beast?

FWEB vs. statement numbers

From: **Allan Adler**
Date: 21 Jun 1994

I am making progress with FWEB (I'm trying the different webs one at a time). As I mentioned in antoher posting, I am practicing by taking a Fortran program I received and converting it to FWEB. At first, most of my effort was spent trying to get fweave to accept it and produce a reasonable TeX file. As soon as I turned to getting ftangle and f77 to accept the web file, all hell broke loose. I managed to fix several mistakes and now it is a lot better. However, I am still having trouble with error messages about multiply defined line numbers. It's like this. One subroutine has something like (in english where possible):

```
13 print a prompt
   get user's input
   if user asks for help, give it and then go to 13
   if user screws up, print .. and go to 97
```

If I set the above as the value of @<Beg the user for information@> and if there is another subroutine that has a statement number 13 beginning another @<@>- module, then I get error messages from f77 about the line numbers being multiply defined (no complaints from ftangle). If you know how to deal with this, please let me know. I placed two files in altdorf.ai.mit.edu:archive/adler/FISCHER for anonymous ftp. The files are named fischer.f and fischer.web. I would appreciate it if someone would look at the original fortran file fischer.f and the web I'm trying to write for it and tell me what I am doing wrong. The trouble all started when web began complaining about line numbers. I didn't put the line numbers in: they were in the code when I got it.

From: **Bart Childs**
Date: 06 Aug 1994

Allan Adler posted a message about debugging an FWEB he was creating from an existing Fortran program. The frustrations he had that caused the posting are similar to those many of us have when we do something that is a little out of the ordinary of our normal mode of operation. I thought I would post this note of how I debugged his problem. I don't think the process is in the FAQ and thought it might help others. Allan posted a second message before I read the first so it might help to know that when I saw it, he had stored both the WEB and the original Fortran.

The steps I took were:

1) I ftp's the files fischer.web and fischer.F (I named it with the uppercase extension because my ftangle would create fischer.f.

2) I ftangle'd fischer.web and produced fischer.f. I did some comparisons of the files fischer.f and fischer.F in a split window in emacs to assure myself that they were somewhat equivalent.

3) I compiled fischer.f on the RS6000 I use and got the messages like his post implied. (Incidentally, I only looked deep enough to make sure of the first message because the others I figured would be similar.) That message was like: ""fischer.f", line 247.1: 1516-034 (E) Statement label is previously defined in this scope. Label is ignored." The offending line of code and some of them around it are:

```
*line 49 "fischer.web"
C* :1 *
C* 22: *
*line 221 "fischer.web"
```

```
C* 23: *
*line 226 "fischer.web"
13    PRINT=.FALSE.    !!!!!!!!!!!This is the one.!!!!!!!!!!!!!!
      SCFTOL=1.D-8
      NSCF=12
      IC=2+(NWF+1-IB)/4
      TRACE=.FALSE.
C* 24: *
*line 240 "fischer.web"
      IF(IB.LE.NWF)THEN
      WRITE(0,'(/A)')' Default values for remaining parameters? (Y/N/H)
     &'
      READ(5,'(A)')ANS
C* 25: *
*line 253 "fischer.web"
```

4) Strings like `PRINT=.FALSE.' may not be so uncommon, but the line before shows `*line 226 ' and I quickly determined that line was in fischer.f twice. Then looking at the INDEX of section names, we see that the code in section 23 (Notice the `C* 23: *' comment above) was ``inserted'' into sections 1 and 22. Hints for that are in the 2nd and 3rd lines of the code excerpt copied. I also can see this information when using web-mode under emacs and issuing the `view-sections' command. It shows the list of numbers of sections that define code for these pseudo-code names and then where they are used.

On unix systems, the following lines of commands (which could easily be a script) can be quite handy. I show them as applied to Allan's file:

```
lanczos ~/fweb> grep -e'@>$' fischer.web | sort > fischer.scn.all
lanczos ~/fweb> grep -e'@>$' fischer.web | sort -u > fischer.scn.uniq
lanczos ~/fweb> diff fischer.scn.all fischer.scn.uniq
```

And the diff step gave this output which is really what is needed.

```
31d30
< @<Determine whether an electron is new or not@>
72d70
< @<Iterate MCHF@>
83d80
< @<Main set parameter default values@>
121d117
< @<Restart the process of setting main defaults@>
```

Thus, Allan had four places where code was copied to two places. The third of these three repeats gives the first compiler error message. I frequently have repeated use of code in Fortran and C. In Fortran, I may have a main and several subprograms that have identical inclusion of COMMONs and similar needs for inclusion in C. A couple of years ago I taught a special topics course on literate programming. After a quick introduction, I gave an assignment of coding the wc.w (unix' word count) from scratch (and luckily none of the students knew where to find it). One student's version worked in spite of some fairly serious errors. (This was done using FWEB) and here is part of the INDEX.tex file that was generated.

```
\Windexspace
\:\\{buf\_ebnd}, \[12].
\:\\{buf\_end}, 13, 16.
\:\\{buf\_size}, 12, 16.
\:\\{buff\_size}, 12.
\:\\{buffer}, \[12], 13, 16.
\:\\{BUFSIZ}, 12.
\:\\{bur\_end}, 16.
```

Due to his errors on the keyboard, he had 3 spellings of buf_end and two of buf_size. Sure, he had some compiler warnings ..., but it worked! Incidentally, I was checking his index to see how many extra entries he had put there (the answer was none). The message is: a quick review of the index of variables (hopefully extra entries too) and index of section names show a lot about your code. Reviewing the latter (or the file fischer.scn.all above) and considering if the names are really appropriate can really help make codes `literate.' I hope this is helpful.

From: **Allan Adler**
Date: 12 Aug 1994

I find that FWEB eliminates spaces in certain pieces of FORTRAN code when I print out the TeX file. For example, something like "CALL ROUTINE" becomes "CALLROUTINE". The same thing happens for declarations. I find that spaces are scrupulously preserved in WRITE statements, however. How do I get fweave to respect the spaces? A similar problem has to do with code that defines one

piece of code as being a bunch of other pieces of code, each being a named module. I would expect to see each of the named modules on a separate line, but for some reason fweave runs them together without line breaks. You can see this in the first few sections of the file fischer.web which is still available at altdorf.ai.mit.edu in archive/adler/FISCHER. I haven't changed the file since I haven't yet figured out which of the duplicated pieces of code I ought to eliminate. Meanwhile, it still serves to exhibit these other problems.

From: **Allan Adler**
Date: 07 Sep 1994

In an earlier message, I complained about FWEB running the words together in statements like "IMPLICIT DOUBLE PRECISION" and "LOGICAL /GLEEP/DUH" I finally found out how to correct it: write everything in lower case. Also, I started reading Knuth's book METAFONT The Program to gain some insight into WEB prose and compositional style. I find this quite helpful and now I feel like I am beginning to get the hang of it. One of the positive features of Knuth's style involves beautifying those parts of the programming language that are usually ugly or hard to read. It is not hard to find such features in Fortran. For example, FORMAT statements are quite ugly and READ and WRITE statements are usually not much better. In addition to these problems, there is also the fact that READ or WRITE statements are often separated from their FORMAT statements by a great distance. I welcome suggestions on how to make these monstrosities beautiful with FWEB.

From: **Bart Childs**
Date: 13 Sep 1994

Allan Adler's comments show that he has realized that FWEB is based on the concepts that UGLY OLD UPPERCASE should be avoided. It does make it a bit more readable. I try to avoid the FORMAT statements as long as it is reasonable. I use constructs like: "read(*,'(8i5)',IOSTAT=io_flag) variable list". The inclusion of the IOSTAT (which needs to be in UPPER in FWEB) makes it a long line which often looks ugly. I have thought of defining macros that would do some of these functions and allow it to be reasonable concise. I hope to have access to an F90 compiler shortly and start trying to avoid the current columnar limitations which make the above problems worse.

FILE and LINE directives

From: **Mark Naumann**
Date: 21 Jun 1994

David Kastrup writes: I recommend switching to a non-pretty-printing Web, such as noweb or nuweb. In my opinion (I have used both CWEB and noweb) the only excuse for using prettyprinting web's is for publication of a program (which has a very special focus on appearance, like typesetting a typesetter source :)), or for fast typesetting of machine-generated code using WEB constructs (if your generated code does not, you are better off with a special purpose prettyprinter like indent), or for writing code where the programmer has as bad a style as to be less readable than machine-formatted code.

For me, pretty-printing is not the reason for using CWEB or FWEB. These tools also automagically create an index showing where each token is used. And, they will insert __FILE__ and __LINE__ directives into the C output when tangling. The later is valuable when debugging. For instance, the CWEB tool will insert file, so that when you are debugging, it puts you into the correct line of your WEB source file. These values are missing from tools that do not know the source language. When using FunnelWeb, we get nice output... but don't immediately/ automagically know which WEB file the C line came from. (The project is large, so there are multiple WEB input files to the documentation.)

From: **Werner**
Date: 21 Jun 1994

Mark Naumann writes: For me, pretty-printing is not the reason for using CWEB or FWEB. These tools also automagically create an index showing where each token is used. And, they will insert __FILE__ and __LINE__ directives into the C output when tangling. The later is valuable when debugging. For instance, the CWEB tool will insert file, so that when you are debugging, it puts you into the correct line of your WEB source file.

You can also try the c2cweb package. It contains a modified CWEAVE which will have a better C support; additionally there will be a distinction between functions and variables in the index (if you have different names).

From: **Norman Ramsey**

Date: 22 Jun 1994

Mark Naumann writes: For me, pretty-printing is not the reason for using CWEB or FWEB. These tools also automagically create an index showing where each token is used.

If you are willing to mark definitions, noweb and nuweb automatically identify uses and create an index. Noweb also creates a mini-index for each chunk, which many programmers have found useful.

And, they will insert __FILE__ and __LINE__ directives into the C output when tangling. The later is valuable when debugging.

This feature is more than valuable; it's indispensable. Noweb and nuweb support C's #line directive, which provides this feature. Noweb also supports the analogous feature for other languages, like Icon and Modula-3. I'm still waiting for a Standard ML compiler supporting this feature...

From: **Lee Wittenberg**
Date: 22 Jun 1994

Mark Naumann writes: For me, pretty-printing is not the reason for using CWEB or FWEB. These tools also automagically create an index showing where each token is used. And, they will insert __FILE__ and __LINE__ directives into the C output when tangling. The later is valuable when debugging. For instance, the CWEB tool will insert file, so that when you are debugging, it puts you into the correct line of your WEB source file. These values are missing from tools that do not know the source language.

This is not exactly true for noweb. It can (and does for C, Icon, and other languages that accept such things) insert #line-like directives (which include the __FILE__ and __LINE__ info). It also can create indices for both identifier and chunk names, the latter automatically, the former with a little help from the programmer (but there exist filters for several languages that let noweb automatically index identifiers, and it is possible to write such a filter for any language you wish).

From: **Tony Coates**
Date: 23 Jun 1994

Mark Naumann writes: For me, pretty-printing is not the reason for using CWEB or FWEB. These tools also automagically create an index showing where each token is used. And, they will insert __FILE__ and __LINE__ directives into the C output when tangling. The later is valuable when debugging. For instance, the CWEB tool will insert file, so that when you are debugging, it puts you into the correct line of your WEB source file. These values are missing from tools that do not know the source language. When using FunnelWeb, we get nice output... but don't immediately/ automagically know which WEB file the C line came from. (The project is large, so there are multiple WEB input files to the documentation.)

If you are interested, you can try my (experimental) version of FunnelWeb, which supports the (non-automatic) insertion of #line directives. I use it all the time myself, and it works perfectly well. However, as far as debugging is concerned, I have found that debuggers choke on #line directives unless the code is laid out sequentially in the original WEB source. For debugging, I replace my FunnelWeb @<line@> macro with a dummy one and thus remove all the #line directives. Yes, I have to recompile, which is a pain, but if I don't the debuggers aren't sure where I am or what I'm doing. That is to say, don't expect debuggers to really support ANSI C.

By the way, my version can create any type of line directive for any language, so long as all that is required is a format containing the line number and file name. So it isn't limited to C/C++. Indeed, I used to use it to imbed comments in Maple files that just made it easier for me to trace errors back to the original FunnelWeb sources.

P.S. Adding #line directives isn't the only way to do things. The version of FunnelWeb supplied with the Eli compiler generation system produces a `map' file which can be used to map compilation errors back to the original FunnelWeb sources. The advantage of *not* using #line directives is that there is nothing to confuse debuggers. I hope to merge the Eli version of FunnelWeb with my own in the near future, with the intention of having it as part of the next official FunnelWeb release.

From: **David Kastrup**
Date: 25 Jun 1994

Mark Naumann writes: For me, pretty-printing is not the reason for using CWEB or FWEB. These tools also automagically create an index showing where each token is used. And, they will insert __FILE__ and __LINE__ directives into the C output when tangling. The later is valuable when debugging. For instance, the CWEB tool will insert file, so that when you are debugging, it puts you into the correct line of your WEB source file. These values are missing from tools that do not know the source language.

You are overgeneralizing. There is no particular reason why a source language independent Web should not make #line directives. Strictly speaking, of course, the form of them does depend on the language in question. But with appropriate options this can be dealt with. Noweb does this. And for debugging purposes, if your compiler/debugger version really chokes up on #line, you can still use the nountangle program, which hides all commentary sections in comments, so you can still see most of what is happening when debugging. Of course, when correcting mistakes, you have to search for the original lines.

When using FunnelWeb, we get nice output... but don't immediately/ automagically know which WEB file the C line came from. (The project is large, so there are multiple WEB input files to the documentation.)

So that is one thing in FunnelWeb that might need amendment.

Tony Coates writes: If you are interested, you can try my (experimental) version of FunnelWeb, which supports the (non-automatic) insertion of #line directives.

You seem to be of the same opinion.

I use it all the time myself, and it works perfectly well. However, as far as debugging is concerned, I have found that debuggers choke on #line directives unless the code is laid out sequentially in the original WEB source. For debugging, I replace my FunnelWeb @<line@> macro with a dummy one and thus remove all the #line directives. Yes, I have to recompile, which is a pain, but if I don't the debuggers aren't sure where I am or what I'm doing. That is to say, don't expect debuggers to really support ANSI C.

Just for interest's sake: What compiler/debugger are you using? Or what combinations have you had these experiences with?

By the way, my version can create any type of line directive for any language, so long as all that is required is a format containing the line number and file name. So it isn't limited to C/C++. Indeed, I used to use it to imbed comments in Maple files that just made it easier for me to trace errors back to the original FunnelWeb sources.

An obvious idea, which noweb supports as well.

From: **Tony Coates**
Date: 29 Jun 1994

David Kastrup writes: You are overgeneralizing. There is no particular reason why a source language independent Web should not make #line directives. Strictly speaking, of course, the form of them does depend on the language in question. But with appropriate options this can be dealt with. Noweb does this. And for debugging purposes, if your compiler/debugger version really chokes up on #line, you can still use the nountangle program, which hides all commentary sections in comments, so you can still see most of what is happening when debugging. Of course, when correcting mistakes, you have to search for the original lines.

I have never used noweb. How does it decide where to insert #line directives? With FunnelWeb, all code sections are macros, and all macros contain code. As such, multiple macros can appear on the one line, and this possibility is what stops me from adding automatic insertion of #line directives with FunnelWeb. I cannot think of an automatic way of knowing which macros will be invoked at the beginning of a line, and not in the middle. Does this differ from the way noweb works? I would be interested to know.

Just for interest's sake: What compiler/debugger are you using? Or what combinations have you had these experiences with?

I have used gdb and DEC's decladebug(ladbx). I tried dbx too. All have failed when the code did not appear sequentially in the FunnelWeb sources. By failed, I mean that they did not report the correct source line. By the way, I was not wishing to suggest that FunnelWeb did anything that some others literate programming tools cannot. The original post mentioned FunnelWeb, and so I answered in that context.

From: **Lee Wittenberg**
Date: 29 Jun 1994

Tony Coates asks: I have never used noweb. How does it decide where to insert #line directives?

Pretty much the same way that CWEB (and the others, I suppose). It simply inserts a #line directive at the beginning of each chunk, and after each expanded chunk. For example, if the following chunk appeared at line 50 of the file xxx.nw:

```
<<*>>=
n++;
<<Some chunk>>
n *= 4;
```

and this chunk appeared at line 273:

```
<<Some chunk>>=
printf("Hello world\n");
```

then the tangled code would look like:

```
#line 50 "xxx.nw"
n++;
#line 273 "xxx.nw"
printf("Hello world\n");
#line 52 "xxx.nw"
n *= 4;
```

Simple, no?

From: **Preston Briggs**
Date: 30 Jun 1994

Tony Coates writes: I have never used noweb. How does it decide where to insert #line directives? With FunnelWeb, all code sections are macros, and all macros contain code. As such, multiple macros can appear on the one line, and this possibility is what stops me from adding automatic insertion of #line directives with FunnelWeb.

In the case of nuweb, the insertion of #line directives causes forces a newline for every macro. Looks ugly, but your code _is_ going to look ugly with all those directives. And it works just fine in dbx, where you'll see only the web source. For example, some expression containing a macro invocation:

```
foo + bar + @ + ...
```

might normally expand into

```
foo + bar + quux
```

With #line directives, we'd see

```
foo + bar +
#line 123 "example.w"
                  quux
#line 1 "example.w"
                       + ...
```

From: **Tony Coates**
Date: 01 Jul 1994

Preston Briggs writes: In the case of nuweb, the insertion of #line directives causes forces a newline for every macro. Looks ugly, but your code _is_ going to look ugly with all those directives. And it works just fine in dbx, where you'll see only the web source. For example, some expression containing a macro invocation:

```
foo + bar + @ + ...
```

might normally expand into

```
foo + bar + quux
```

With #line directives, we'd see

```
foo + bar +
#line 123 "example.w"
```

```
                    quux
#line 1 "example.w"
                        + ...
```

OK, I can see that this would be reasonable in most circumstances, though I wouldn't make FunnelWeb work this way myself. Why? Because in FunnelWeb, I can do things like

```
@$@<String 1@>==@{The rain in Spain@}
@$@<String 2@>==@{falls mainly in the plain.@}
```

and then stick these definitions in the one string as

```
@$@<Full String@>==@{"@<String 1@> @<String 2@>@"}
```

The compiler would probably choke on a line directive in a string. Not that I expect this to happen often, but equally there's no reason why someone shouldn't do it (I know that in ANSI C I might be able to get around it by splitting the string over two lines, but I don't know what the behaviour is if the two strings are separated by a #line directive. It should *probably* work, but I wouldn't trust all compilers to agree on that).

By the way, I certainly have had problems debugging with dbx (on a DEC Alpha). I'm not sure whether the problem was caused primarily by the code being out of order in the FunnelWeb source files, or by some routines having components from different files, or by the fact that some macros are used multiple times. So if anyone *hasn't* had problems, I wonder if that isn't more through good luck than through having a debugger that genuinely can handle arbitrary #line directives in the original sources.

My personal feelings is that a separate map file is the most flexible alternative, though it does complicate the issue by requiring translators for the compiler error reports and for the debugger source line/file position indication. I'm not yet convinced that the #line directive way of doing things is yet flexible enough, much as it works well in a very great number of cases.

From: **Marc van Leeuwen**
Date: 01 Jul 1994

Tony Coates writes: OK, I can see that this would be reasonable in most circumstances, though I wouldn't make FunnelWeb work this way myself. Why? Because in FunnelWeb, I can do things like

```
@$@<String 1@>==@{The rain in Spain@}
@$@<String 2@>==@{falls mainly in the plain.@}
```

and then stick these definitions in the one string as

```
@$@<Full String@>==@{"@<String 1@> @<String 2@>@"}
```

Are you sure you dont mean ... @<String 2@>"@} there?

*The compiler would probably choke on a line directive in a string. Not that I expect this to happen often, but equally there's no reason why someone shouldn't do it (I know that in ANSI C I might be able to get around it by splitting the string over two lines, but I don't know what the behaviour is if the two strings are separated by a #line directive. It should *probably* work, but I wouldn't trust all compilers to agree on that).*

Not knowing FunnelWeb too well I can't see whether this is explicit design or just consequence of the general language independent setup. In any case it wouldn't be possible in CWEB because strings are single tokens there, so at best such code would yield a string with "String 1" and "String 2" as substrings rather than "Spain" and "plain" (in fact the undoubled @-signs in the string would trigger error messages). You could get the intended effect using the (ANSI) C string break by writing

```
@ @<String 1@>== "The rain in Spain"
@ @<String 2@>== "falls mainly in the plain."
@ @<Full String@>== @<String 1@> " " @<String 2@>
```

which CTANGLEs to something like

```
/*4:*/
#line 23 "src.w"
/*2:*/
#line 21 "src.w"
```

```
"The rain in Spain"
/*:2*/
#line 23 "src.w"
" "/*3:*/
#line 22 "src.w"
"falls mainly in the plain."
/*:3*/
#line 23 "src.w"
/*:4*/
```

Despite all the intervening stuff the three string fragments get properly concatenated by the compiler, and since the order of transformations in the C preprocessor is quite strictly specified, this should not depend on the compiler used.

*I'm not sure whether the problem was caused primarily by the code being out of order in the FunnelWeb source files, or by some routines having components from different files, or by the fact that some macros are used multiple times. So if anyone *hasn't* had problems, I wonder if that isn't more through good luck than through having a debugger that genuinely can handle arbitrary #line directives in the original sources.*

My experience with #line directives introduced by CWEB is that they usually work like a charm with source code debuggers (I've used both gdb and a turbo C debugger), except that sometimes you may briefly step to improper lines (e.g., documentation lines) after which you quickly return to the right place; I think this may be due to optimizations which make the correspondence of source code to compiled code not one-to-one, with the confusion being compunded by the many #line directives. In principle a debugger should have no more difficulty with #line directives then a compiler, and in fact it is probably not even aware of their presence: the compiler will have inserted source line indications into the compiled code, and the debugger cannot know whether these are truthful or have been influenced by #line directives.

In one case there is a genuine problem, if the same source line is used multiply in the compiled code, namely when the debugger has to translate in the opposite direction, from source line to compiled code location. Suppose a source chunk is used three times and you decide to put a breakpoint in the chunk; what does the debugger do? This is really a general problem of the debugger, not of the producer of the #line directives; indeed it can occur even without those directives if a file is multiply #included. Chances are that only one occurrence of the source line really gets a breakpoint, probably the first or last occurrence. This would be inconvenient, but the situation is so rare that I never experienced it in practice.

From: **Preston Briggs**
Date: 01 Jul 1994

Tony Coates writes: OK, I can see that this would be reasonable in most circumstances, though I wouldn't make FunnelWeb work this way myself. Why? Because in FunnelWeb, I can do things like

```
@$@<String 1@>==@{The rain in Spain@}
@$@<String 2@>==@{falls mainly in the plain.@}
```

and then stick these definitions in the one string as

```
@$@<Full String@>==@{"@<String 1@> @<String 2@>@"}
```

Sure, these things are also possible in nuweb and noweb. The problem actually came up once here, so I can't argue that people don't ever want to do it. On the other hand, it's awfully easy to get around; hardly worth giving up nice error messages and source-level debugging. As I recall, the problem that came up here was slightly simpler, along the lines of:

```
        fprintf(stderr, "@<Error message@>\n");
```

and

```
        @d Error message @{This'll break Preston's code!@}
```

How to define pretty-printing for variables like "N"?

From: **Matthew Pharr**
Date: 12 Jul 1994

Having discovered the fun of getting variables named omega, say, be printed out with real Greek symbols when my programs are woven and printed, I'm now trying to get arrows over my vector-based variables. This works just fine for variables named, for example, "Ng":

```
@f Ng TeX
\def\Ng{\vec{Ng}}
```

but most of the upper-case versions of the letters in the alphabet have already been taken for other uses in cwebmac.tex, so I can't do the same with vectors named "N", or "V". For now, I've renamed my variables in my program (e.g. N became Nvec, \def\Nvec{\vec{N}}), but I'd really like to be able to have my arrows over my vectors without renaming the variables--is this possible, or am I out of luck?

From: **Lee Wittenberg**
Date: 14 Jul 1994

Matt Pharr asks: Having discovered the fun of getting variables named omega, say, be printed out with real Greek symbols when my programs are woven and printed, I'm now trying to get arrows over my vector-based variables. This works just fine for variables named, for example, "Ng":

```
@f Ng TeX
\def\Ng{\vec{Ng}}
```

but most of the upper-case versions of the letters in the alphabet have already been taken for other uses in cwebmac.tex, so I can't do the same with vectors named "N", or "V". For now, I've renamed my variables in my program (e.g. N became Nvec, \def\Nvec{\vec{N}}), but I'd really like to be able to have my arrows over my vectors without renaming the variables--is this possible, or am I out of luck?

As best I can figure it, you are out of luck here, but single-letter variable names are a Bad Thing in literate programs for the same reasons that they cause trouble in normal programs. The human programmer modifying your .w file may not read N correctly (and IMO most people, when they see a typeset \vec{N} say "vector N", or "N vector", or something like that to themselves when they read it, so Nvec is much better than plain N).

Operator overloading in CWEB

From: **Greg Fiehler**
Date: 12 Jul 1994

I have written a few CWEB programs with C, but I am just learning C++ and I am trying to write some of my code in CWEB and I am running into a couple of problems. These are mainly formatting problems and may just be my lack of TeX experience but I would appreciate any comments.

My first problem is with overloading operators, when I write a section definition as @<operator& member function@>@; the & confuses things. I also have the same problem with the | operator but this is even worse. I have a section titled "@ Overloading the | operator." The parser looks for a second | so that it can put the text in between in the C style format, I have tried to `\'escape the | char. and I have tried using $|$ hoping TeX's math mode would help, but both were unsuccessful.

Also I am not sure I like CWEB's practice of changing the ! to the math symbol and the ^ to the xor math symbol and the == to the triple equal (although that is not as bad as the others) to me this is confusing if the purpose of the output is to make the program more readable for program upkeep, or for the purpose of documenting a class so others may utilize it. In C it was ok, but when I overload an operator I think it would be more readable for a programmer if the symbol was left intact.

Last I sometimes like to put comments in between code segments when they are different but not enough to warrant thier own section CWEB wants to put them after a line of code and not on the next line, i.e. I have tried the @; , @/ , and @#. None of these seem to work.

```
code ...
/* some comments */
code ...

is formatted as:
```

```
code... /* some comments */
code...
```

My last problem is on designing a large CWEB program. I will be starting on some large projects soon and would like to use literate programming but I am not sure how to organize things. First I would normally use separate files for at least groups of like functions if not separate for each function. I know how to have my web file write the tangled code to separate files but my problem is in how do you manage a web file for a program that would normally take over 10,000 lines of code. And if you keep everything in one web file unless I am mistaken on how `make, works each time I make a revision every file will be compiled again instead of just the one in which the change was needed. I would appreciate any comments on how to handle these prooblems.

From: **Stephen Fulling**
Date: 13 Jul 1994

Greg Fiehler writes: I have a section titled "@ Overloading the | operator." The parser looks for a second | so that it can put the text in between in the C style format, I have tried to `\'escape the | char. and I have tried using $|$ hoping TeX's math mode would help, but both were unsuccessful.

Try \vert or $\vert$ (depending on context).

From: **Lee Wittenberg**
Date: 14 Jul 1994

Gregg Fiehler writes: My first problem is with overloading operators, when I write a section definition as @<operator& member function@>@; the & confuses things.

There are 2 ways to deal with this (depending on what you're trying to accomplish). Use @<operator\& member function@> or @<|operator&| member function@>

I also have the same problem with the | operator but this is even worse. I I have a section titled "@ Overloading the | operator."

Stephen Fulling's suggestion of \vert or $\vert$ is probably the best solution here.

Also I am not sure I like CWEB's practice of changing the ! to the math symbol and the ^ to the xor math symbol and the == to the triple equal (although that is not as bad as the others) to me this is confusing if the purpose of the output is to make the program more readable for program upkeep, or for the purpose of documenting a class so others may utilize it. In C it was ok, but when I overload an operator I think it would be more readable for a programmer if the symbol was left intact.

Personally, I find, over time, that I much prefer the more mathematical symbols as a nearly language-independent "publication language" (as WEB, FWEB, and many Spidery webs print the same symbols for the same operations, even though the input languages are different). However, if you prefer something different, you can always redefine \R, \XOR, \E, etc. in your copy of cwebmac.tex (that way others -- such as me -- can typeset your webs using the standard symbols or whatever we find most readable). If you want to mix C & C++, you might also want to play around with a custom \if that allows you to specify whether segments are typeset normally or with your own versions.

Yet another technique (trick?) that you might find useful in this situation is to use @d's and @f's in combination. For example, I sometimes like to be able to differentiate between multiplication and the pointer dereference operator (`*'). The declarations "@d times *", "@f times TeX" do the trick (as \times is already defined). Then I can type "x = y times z;" in my web and it will be typeset in TeX as $x \gets y \times z;$ (and the preprocessor will take care of turning the tangled `times' into `*'.

Last I sometimes like to put comments in between code segments when they are different but not enough to warrant thier own section CWEB wants to put them after a line of code and not on the next line, i.e. I have tried the @; , @/ , and @#. None of these seem to work.

```
code ...
/* some comments */
code ...

is formatted as:

code... /* some comments */
code...
```

There may be an easier way, but this should do the trick. Substitute \7 for \6 to achieve the effect of @# and you can use \4 or \8 (after the \6 or \7) to make the comment stick out to the left.

```
code ...
@t}\6{@>
/* some comments */
code ...
```

My last problem is on designing a large CWEB program. I will be starting on some large projects soon and would like to use literate programming but I am not sure how to organize things. First I would normally use separate files for at least groups of like functions if not separate for each function. I know how to have my web file write the tangled code to separate files but my problem is in how do you manage a web file for a program that would normally take over 10,000 lines of code. And if you keep everything in one web file unless I am mistaken on how `make, works each time I make a revision every file will be compiled again instead of just the one in which the change was needed.

Judicious use of @i (along with appropriately modified dependencies in the makefile) may do the job. On the other hand, I find that the human time I save by using literate programming tools (even on large programs) more than outweighs the extra processor time involved even when I keep the entire web in a single .w file. And human time, after all, is more expensive (and precious) than computer time.

From: **Brian Stuart**
Date: 14 Jul 1994

Greg Fiehler writes: My last problem is on designing a large CWEB program. I will be starting on some large projects soon and would like to use literate programming but I am not sure how to organize things. First I would normally use separate files for at least groups of like functions if not separate for each function. I know how to have my web file write the tangled code to separate files but my problem is in how do you manage a web file for a program that would normally take over 10,000 lines of code. And if you keep everything in one web file unless I am mistaken on how `make, works each time I make a revision every file will be compiled again instead of just the one in which the change was needed.

What I've been doing on a current project is to keep separate .w and .hw (for the header files) files just as I would normally with .c and .h files. That way make can tangle and compile only those that have been changed or that depend on changed files. For weaving, I use a .w file the includes (with the @i operator) each of the .w and .hw files. This top-level file also contains the copyright notice that appears in the printed version as well as an overall description of the library or whatever. Below are a sample Makefile and a top-level .w for weaving. (Obviously the Makefile could use comments and the .w could use a little more overall description, but my defense is that it's a work in progress? :-)

As long as we're at it, I also have a question regarding this and similar projects. The bulk of the project code consists of a number (around 30) different programs each (currently) in single files. They are also weaved as a set. The problem is that many of the programs use the same variable names and the index is nearly useless. On the other hand, I really don't want a separate index and table of contents for each program in the final document. (I'm thinking about publishing it. It's a real-world small business management system produced under an unsusal contract whereby I retain the rights to the code.) What I'd really like is to divide a common index into sections, one for each program. Has anyone else tried to deal with this type of situation? How did you resolve it? I haven't picked up a copy of The Stanford GraphBase yet, but the code I pulled off the net appears to use the separate toc and index approach. Now I just have to figure a good way to do Lex code in CWEB... Anyway, here's the Makefile and librec.w.

```
--- Makefile ---
.SUFFIXES: .w .hw .tex .dvi

CC = gcc
INCFLAGS = -I.
CFLAGS = $(INCFLAGS) -O

WEBS = record_io.hw lock.w opcl.w read.w write.w apply.w search.w \
        delete.w num_rec.w
SRCS = opcl.c lock.c apply.c search.c delete.c num_rec.c read.c write.c
OBJS = opcl.o lock.o apply.o search.o delete.o num_rec.o read.o write.o

.hw.h:
        ctangle -bhp $< - $@

.w.c:
```

```
	ctangle -bhp $<

.w.tex:
	cweave -bhp $<

.tex.dvi:
	tex $<

librec.a: $(OBJS)
	ar rv librec.a $(OBJS)
	ranlib librec.a

test_rec: test_rec.o librec.a
	$(CC) $(CFLAGS) -o test_rec test_rec.o librec.a

read_all: read_all.o librec.a
	$(CC) $(CFLAGS) -o read_all read_all.o librec.a

test_block: test_block.o librec.a
	$(CC) $(CFLAGS) -o test_block test_block.o librec.a

reader: reader.o librec.a
	$(CC) $(CFLAGS) -o reader reader.o librec.a

writer: writer.o librec.a
	$(CC) $(CFLAGS) -o writer writer.o librec.a

test_rec.o: test_rec.c record_io.h
read_all.o: read_all.c record_io.h
test_block.o: test_block.c record_io.h
reader.o: reader.c record_io.h
writer.o: writer.c record_io.h

install: librec.a
	cp librec.a ../../lib
	cp record_io.h ../../include

clean:
	rm -f *.o *.dvi *.log *.idx *.scn *.toc core

opcl.o: opcl.c record_io.h
lock.o: lock.c record_io.h
apply.o: apply.c record_io.h
search.o: search.c record_io.h
delete.o: delete.c record_io.h
num_rec.o: num_rec.c record_io.h
read.o: read.c record_io.h
write.o: write.c record_io.h
librec.tex: librec.w $(WEBS)

--- librec.w ---

\def\title{Appendix A: Record I/O Library}
@**Record I/O Library.

@ All code written by Brian L. Stuart.

@ Copyright 1993, 1994 Brian L. Stuart

@s delete x

@ The code in this web makes up a record I/O library to be used
in the Gin Management System.
It provides basic opening, closing reading and writing access to
files of records.
One simplifying assumption is made; each file consists only of
a series of like-typed records.
```

```
In addition to the basic operations, this library provides routines for
searching a file for a record that meets some criterion and
for deleting records that meet some criterion.
Futhermore, two forms of |apply()| are provided that allow for the
application of some function to all records in a file.

@ Locking is provided to manage multiple processes that access the
database simultaneously.
The locking is done on a per-file basis.
If our system supports file locking based on the |fcntl()| system call,
then we allow concurrent readers.
Otherwise, we use lock files and all locks are exclusive.

@i record_io.hw
@i lock.w
@i opcl.w
@i read.w
@i write.w
@i num_rec.w
@i apply.w
@i search.w
@i delete.w

@**Index.
```

From: **Lee Wittenberg**
Date: 15 Jul 1994

Brian Stuart writes: As long as we're at it, I also have a question regarding this and similar projects. The bulk of the project code consists of a number (around 30) different programs each (currently) in single files. They are also weaved as a set. The problem is that many of the programs use the same variable names and the index is nearly useless. On the other hand, I really don't want a separate index and table of contents for each program in the final document. (I'm thinking about publishing it. It's a real-world small business management system produced under an unsusal contract whereby I retain the rights to the code.) What I'd really like is to divide a common index into sections, one for each program. Has anyone else tried to deal with this type of situation? How did you resolve it? I haven't picked up a copy of The Stanford GraphBase yet, but the code I pulled off the net appears to use the separate toc and index approach.

The published Graphbase has a single index for all the programs with uses sorted by original filename (e.g., date: FOOTBALL 16, GB_GAMES 5, 24.) I suspect that Knuth hacked CWEAVE (the |phase_three| function, section 225 in CWEAVE 3.0) and did a kind of merge-sort on the resulting indices before TeXing the lot.

Now I just have to figure a good way to do Lex code in CWEB...

Good luck. For mixed languages, I usually end up using noweb (or a heckuva lot of @='s for the non-C/C++ code).

From: **Marc van Leeuwen**
Date: 15 Jul 1994

Greg Fiehler writes: My first problem is with overloading operators, when I write a section definition as @<operator& member function@>@; the & confuses things. I also have the same problem with the | operator but this is even worse. I have a section titled "@ Overloading the | operator." The parser looks for a second | so that it can put the text in between in the C style format, I have tried to `\'escape the | char. and I have tried using $|$ hoping TeX's math mode would help, but both were unsuccessful.

The problem with `&' is that the cwebmac format uses `\&' for its own purposes, so you can use it like in plain TeX. However, the same format gives you `\AND' which it uses to produce the bitwise-and and the address-of operators, and which the programmer can use inside math mode, if desired; you could therefore write `@< operator$\AND$ member funtion @>'. If you prefer the (non-math) function of the traditional `\&', you could say in limbo something like "\chardef\amp=`\&" and use \amp where you would otherwise use \&, e.g., `@< operator\amp\ member funtion @>'.

The problem with `|' is a bit more fundamental, since CWEAVE will pick up (nearly) all occurrences of `|' as C-mode delimiters even before TeX ever gets to see them. An exception is between `@t' and `@>', so that (by a trick I learned from Joachim Schrod) you can say "@ Overloading the `|@t$ | $@>|' operator." (note that from the outside inwards we change from text mode to C-mode to text

mode to math mode here, since `@t' requires C-mode and `|' requires math mode. This trick will not work in module names like you needed for the operator& example though, since `@t ... @>' cannot be used there. However, there is \OR which is much like \AND except that the symbol is of course `|' rather then `&' (actually \OR is just another name for \mid) and you can define "\chardef\bar=`\|" in limbo if you like, but remember that there is no vertical bar in the ordinary text fonts like cmr10 or cmbx10, so you could use this character only with \tt fonts.

Also I am not sure I like CWEB's practice of changing the ! to the math symbol and the ^ to the xor math symbol and the == to the triple equal (although that is not as bad as the others) to me this is confusing if the purpose of the output is to make the program more readable for program upkeep, or for the purpose of documenting a class so others may utilize it. In C it was ok, but when I overload an operator I think it would be more readable for a programmer if the symbol was left intact.

That is just a matter of taste (personally I render the == as an ordinary math equals and the assignment operator as a \Leftarrow, which has helped me catch many "comparisons by assignment operator" typos). In any case you can redefine macros of cwebmac to suit you taste. The relevant ones for you are \R for `!', \XOR for `^' and \E for `==='; other operators can be found by looking at the cwebmac.tex file or (easier) by a simple experiment.

Last I sometimes like to put comments in between code segments when they are different but not enough to warrant thier own section CWEB wants to put them after a line of code and not on the next line, i.e. I have tried the @; , @/ , and @#. None of these seem to work.

```
code ...
/* some comments */
code ...

is formatted as:
code... /* some comments */
code...
```

As far as I know, you are out of luck here, since CWEAVE insists on putting comments to the left of something, discarding any breaks that may precede it. You can try to be devious though and attach the comment to a dummy and invisible statement created by @t@> @; and preceed that by a forced break but I'm not sure you will like the spacing.

From: **Richard Walker**
Date: 22 Aug 1994

I have searched the CWEB manual, but I can't find the answers to these questions. Can you help? 1. How do I insert a literal vertical bar (|) in the documentation part? Do I have to use \char"7c ? If so, what do I do if I am using a verbatim environment? 2. How do I insert a vertical bar in the middle of C code surrounded by | ... | ?

From: **Marc van Leeuwen**
Date: 22 Aug 1994

Richard Walker writes: I have searched the CWEB manual, but I can't find the answers to these questions. Can you help? 1. How do I insert a literal vertical bar (|) in the documentation part? Do I have to use \char"7c ? If so, what do I do if I am using a verbatim environment?

You can say \chardef\v=`| in the limbo part so that you can write `\v' instead of `\char"7c'. Note however that the ordinary text fonts don't have a vertical bar (you would get `---' instead), so it would only work with fonts like \tt (and in math mode). You shouldn't be using a verbatim environment though, since, as with any system using a preprocessor (in this case CWEAVE), it is impossible to create a true verbatim environment after the preprocessing, for the simple reason that original text simply isn't there any more. This is assuming the preprocessor, like CWEAVE, does not itself have a verbatim mode; even if it would, managing the various verbatim modes simultaneously would be quite tedious.

2. How do I insert a vertical bar in the middle of C code surrounded by | ... | ?

There is no easy way to mention the bitwise-OR operator `|' within `|...|' in CWEB. You can write `@t$|$@>' within `|...|' to obtain the desired symbol, but it would not be parsed as an operator by CWEAVE, so the output might still look wrong. If your expression is not too complicated, you could use `$...$' instead of `|...|', and then use either the plain TeX symbol `\vert', or `\v' as defined above.

From: **Brian Stuart**
Date: 20 Dec 1994

A while back, I asked how people were formatting systems of software with many webs. After discussing it a little, I got into one of those challenged moods and decided to create a little script for taking several index files and combining them into one index file where each entry would be tagged with the source filename. At first I thought about writing a program to do this, but the more I thought about it the more I thought that this was exactly what software tools like sed, awk, sort, join, etc were for. So I was just crazy enough to do it. Originially, I had intended to do one for the sections as well, but never decided how it should look. Since it looks like it'll be a while before I get around to that, here's the hack for the index files. You can also find it in: ftp://mathcs.rhodes.edu/pub/literate/comm_idx.sh

```
#!/bin/sh
#
# Combine the index files from several CWEBs into one.
# by Brian L. Stuart
# A script for the truly masochistic.
#
# This script may be freely distributed, modified, scavenged, folded
# spindled or mutliated provided that appropriate credit/blame of
# original authorship is given.
#
# Absolutely no warantees on this one, folks.
#
# Example Usage:
#    comm_idx a.idx b.idx ... > big.idx
#
# We'll do all our work in /tmp.
#
# We'll assume that the section numbers in the index files are what
# you want. That means that if you want the sections numbered
# consequectively across file boundaries, you need to make sure that
# that happens when you generate the index file.
#
# One other caveat. If join or sort have problems with excessively
# long lines, this may crash. The GNU versions seem to be able to
# handle lines long enough to deal with the Stanford GraphBase.
# Also non-GNU versions of join may not grok the use of standard
# input for the second filename. There is also some question about
# whether or not other versions of join can handle both -a1 and -a2
# options.
#
rm -f /tmp/$$.total
touch /tmp/$$.total
#
# The input index files are listed as command line arguments with
# the idx extensions present.
#
for i do
   #
   # Echo the index file name to the standard output so the user can
   # keep up with it's progress
   #
   echo $i >&2
   #
   # The next line removes the .idx extension and replaces any
   # _ (underscores) with \\_
   #
   NAME=`echo $i | sed -e 's/\\(.*\\)\\.idx/\1/' -e 's/_/\\\\\\\\_/g'`
   #
   # Now for the truly scarey part. For each index file:
   #   1. Append any lines that are part of the same index entry. Any
   #      lines that don't end in `.' should have the next line appended.
   #   2. Replace any \9s with \#. (The later sort will think \9 should
   #      be part of the sort key and put all them first.)
   #   3. Replace any commas that don't mark the end of the index entry
   #      name with @. The join will get confused by any commas
   #      that appear in the index entry string (like for names).
   #   4. Sort the input file in ASCII order (different from the
```

```
  #      dictionary order they're in.
  #   5. Put the root file name after each index entry name.
  #   6. Join this modified file to the one we're accumulating.
  #
  sed -e :1 -e N -e 's/\([^.]\)\n/\1 /' -e t1 \
     -e 's/\\9/\\#/' -e 's/\([^}]\),/\1@/g' \
     -e P -e 's/.*\n//' -e b1 < $i | \
     sort | \
     #
     # If you prefer the file names in italics, include this next
     # line and comment out the following one. The second one
     # prints the file names in roman type.
     #
     # sed "s/\\(.*},\\)\\(.*\\)/\\1 \\\\\\\\{$NAME},\\2/" | \
     sed "s/\\(.*},\\)\\(.*\\)/\\1 $NAME,\\2/" | \
     join -a1 -a2 -t, /tmp/$$.total - > /tmp/$$.new
  mv /tmp/$$.new /tmp/$$.total
done
#
# The new file we've accumulated is sorted in ASCII order and we need
# it in dictionary order. We also need to undo the changes to the commas
# and the \9s. Before those changes, however, we also fold the output
# so that TeX doesn't complain about line lengths.
#
sort -df /tmp/$$.total | fold -s | sed -e 's/\\#/\\9/' -e 's/@/,/g'
rm /tmp/$$.total
```

Bold identifiers and plain keywords

From: **John Scholes**
Date: 15 Jul 1994

Has anyone had any experience of literate programming with their normal standard editor (rather than TeX etc). I use C++, but I don't like the Knuth single character substitutions (an arrow for -> etc). On the other hand, I quite like the idea of making keywords bold. I also wanted a product that will strip out the non-code part and produce a pretty printed result, so that the it can be used by those who don't like literate programming. Any practical experience anyone?

From: **Daniel Simmons**
Date: 17 Jul 1994

John Scholes writes: I use C++, but I don't like the Knuth single character substitutions (an arrow for -> etc). On the other hand, I quite like the idea of making keywords bold. I also wanted a product that will strip out the non-code part and produce a pretty printed result, so that the it can be used by those who don't like literate programming. Any practical experience anyone?

Well, I've been using FunnelWeb because I write programs in a variety of languages, and (more importantly) I write *systems* that have programs in multiple languages. Another advantage of FunnelWeb, though, is that it doesn't pretty print the code at all. Instead, it gives you 100% control over your code output. So my literate programs are designed to be readable in both woven and tangled forms. I put comments in my code sections as well as around them. It seems to work fairly well.

From: **Lee Wittenberg**
Date: 17 Jul 1994

John Scholes asks: Has anyone had any experience of literate programming with their normal standard editor (rather than TeX etc). I use C++, but I don't like the Knuth single character substitutions (an arrow for -> etc). On the other hand, I quite like the idea of making keywords bold. I also wanted a product that will strip out the non-code part and produce a pretty printed result, so that the it can be used by those who don't like literate programming. Any practical experience anyone?

noweb is designed to allow for just such experimentation. There are several filters available that pretty-print code, and there are at least 2 back-ends (one for TeX, the other for HTML). You can also write your own back-end for whatever editor or formatter you like (a troff back-end would be a great service to many, I suspect). You might also want to look at CLiP and WinWordWEB.

From: **John Hamer**
Date: 19 Jul 1994

I use C++, but I don't like the Knuth single character substitutions (an arrow for -> etc). On the other hand, I quite like the idea of making keywords bold.

Ever thought of making identifiers bold, and leaving keywords alone? It seems odd at first, but when the novelty wears off you'll wonder why anyone would have it any other way. The idea of emphasising identifiers at the expense of keywords comes from Ledgard and Tauer's book "Professional Software, Volume II: Programming Practice", Addison-Wesley (1987) ISBN: 0-201-12232-4. This is well worth a read.

From: **Barry Schwartz**
Date: 19 Jul 1994

I also wanted a product that will strip out the non-code part and produce a pretty printed result, so that the it can be used by those who don't like literate programming.

Well, the best thing would be to shun those people. But, seriously, you could try running the C output through indent or tgrind. You'd lose the documentation, but if they really wanted the documentation they wouldn't dislike literate programs.

From: **Stephen Fulling**
Date: 19 Jul 1994

John Hamer writes: Ever thought of making identifiers bold, and leaving keywords alone? It seems odd at first, but when the novelty wears off you'll wonder why anyone would have it any other way. The idea of emphasising identifiers at the expense of keywords comes from Ledgard and Tauer's book "Professional Software, Volume II: Programming Practice", Addison-Wesley (1987) ISBN: 0-201-12232-4. This is well worth a read.

I've said this before, but it bears repeating. The standard practice of using E X T E N D E D bold for keywords and the cramped italic font for identifiers puts the emphasis in the wrong place. I suspect it contributes to many people's lack of enthusiasm for pretty-printing. I didn't particularly like the result of interchanging the two fonts -- and other people I showed it to liked it even less. So I settled on a compromise that seems near optimal: Use sans-serif for keywords and slanted roman for identifiers. This is easily done by a few changes to cwebmac.tex The slanted font is larger and easier to read than italic. The sans-serif is less obtrusive than bold-extended and makes a better contrast with \tt (used for strings).

From: **David Kastrup**
Date: 20 Jul 1994

John Scholes writes: I also wanted a product that will strip out the non-code part and produce a pretty printed result, so that the it can be used by those who don't like literate programming.

Barry Schwartz writes: Well, the best thing would be to shun those people. But, seriously, you could try running the C output through indent or tgrind. You'd lose the documentation, but if they really wanted the documentation they wouldn't dislike literate programs.

Another option is to use noweb. When tangling without #line information, it will correctly preserve indentation of the various modules. There is even a program nountangle with it, which will turn the documentation parts into comments of the tangled language. Very neat. Also, the line numbers in the weaved file are the same as in the web source, making TeX error messages pinpoint the right numbers.

Anyone use ProTex?

From: **Sean Boyle**
Date: 29 Jul 1994

I downloaded it, untexar'd it (TeX archive -vs- shell archive... cute), and found only one example of the literate programming stuff. It is

short, uncommented and doesn't get me very far. I have the book on order, but I confess that I feel a bit silly buying a book on a product I know nothing about, just to see what it can do. It looks like it comes with a bunch of macros to do charts and graphics more easily, but no docs on them either, it must all be in the book. In short, does anyone have any experience with it?

From: **Stephen Fulling**
Date: 30 Jul 1994

I hope that Eitan Gurari will speak for himself, but here is my limited input. I have read the relevant chapters of the book but have not seriously tried out ProTex yet. I think it deserves to be taken seriously. Its strong points are PORTABILITY and simplicity: It is built entirely out of TeX macros, so there is no need for extra programs (like weave and tangle) or shell scripts, etc., that must be different for Unix and DOS (as with Noweb). Any computer equipped with TeX can run ProTeX. Its main disadvantage, as I understand it, is weak or slow indexing capability.

Also, you get the (completely independent) DraTeX graphics macros as part of the deal. In fact, most of the book is about them; the literate programming is covered in just a few chapters at the end. I have not tried them or even read that part of the book. The file is considerably shorter than PicTeX, so it may solve some of the memory problems that curse PicTeX on many systems. But do NOT buy the book just to get the software. The files available for free by FTP are more up to date than those on the diskette included with the book.

A parsing bug in CWEAVE?

From: **Felix Gartner**
Date: 02 Aug 1994

I have ve come across a situation where the CWEAVE parser doesn't seem to do the job correctly. Has anybody else discovered this and is there are solution to the problem? Anyway TeX takes care of the problem but it's not nice that CWEAVE can't handle this on its own.

```
---------example follows: ----------------------------------------

@ Try this:

@c
#define string char*    /* why doesn't this work? */

----------end example-----------------------------------------------
```

From: **Roland Kaufmann**
Date: 03 Aug 1994

With CWEAVE 2.1 (yes, I know it's old ;-), the following workaround solves the problem:

```
@ Try this:

@d string @[char*@]     /* It's ugly, but it works. */
```

but the more natural (? to a C programmer) variant

```
@ Try this:

@c
#define string @[char*@]        /* This is going to disappear from the listing! */
```

is even worse than the original: the entire code is lost from the listing (i.e. TeX file), although it is tangled fine. Has anyone tried a more recent version of CWEAVE?

From: **Lee Wittenberg**
Date: 03 Aug 1994

Werner Lemberg and Roland Kaufmann have proposed solutions, but might I suggest a different tack? Try

```
@c
typedef char *string;
```

#defines used in this situation can cause strange (and hard to find) problems. An added advantage is that CWEB will now recognize string as a type name, and typeset it like `int'.

From: **Barry Schwartz**
Date: 03 Aug 1994

You don't say what the problem is. The real solution is to use a typedef rather than a macro. If you can't use a typedef, then you are probably doing something you shouldn't be doing. But if you just want to know how to get the correct formatting, try adding "@f string int" before the @c.

From: **Marc van Leeuwen**
Date: 04 Aug 1994

The syntax used by (Levy/Knuth) CWEB is so complicated that I would be very surprised if it worked at all reliably (i.e., leading to proper formatting) in strange situations. However, in this particular case it produces code that makes TeX choke, which should of course not happen. Schematically the problematic output looks like

```
$ ... math stuff ... { ... $ \C{ comment } $ ... } ... $
```

The $'s were put around the comment since it should be processed in text mode, but the extra braces around it cause TeX to protest about this way of temporarily getting out of math mode. The extra braces are caused by rule 34, which attemts to put braces around a `*' to prevent it from becoming a binary opertor. However rule 0 has already stuck the comment onto the `*', so the braces go around it as well. One cannot change the definition of \C using \hbox so that it will survive in math mode, since comments must be allowed to break across lines. So I would say that rule 34 (in combination with the general rule 0) is fundamentally flawed, as are other rules that stick in braces. Here is another problematic case (by rule 18)

```
@c
* p = * /* why would one want to put a comment here? */ q;
```

By the way, if your example had been with `@d' instead of `#define', it would even fail without the trailing comment, since there is insert-like stuff after a `@d' definition.

Has anybody else discovered this and is there are solution to the problem?

I'd say the only way is to avoid using braces that are not immediately matched (without intervening scraps) in the RHS of rules altogether. I have done this in my version of CWEB that is available from the LPA.

From: **Andreas Scherer**
Date: 04 Aug 1994

There have been several different answers to this problem, the best of which are by Lee Wittenberg and Barry Schwartz, who both suggest using "typedef" instead of "define". I agree. But if you really absolutely have to use the construction above, use it in the following form:

```
@ Try this

@s string int

@c
#define string char* @; /* `@;' is the clue! */
```

This works in every case with a "type name" as replacement text. And if you look at the analysis mode `@1', it is often advisable to put an extra `@;' at the end of the line, especially with tricky constructions in parametric macros. Even in C-itations `@;' has its benefits. (In the case above there is an irreducible `decl' scrap left, but this is better than a `decl_head' without the `@;', because this also gives some problems to TeX.)

Arguments for literate programming

From: **Denis Roegel**
Date: 31 Aug 1994

I was trying to convince somebody to use CWEB for his C programming. As usual, people are skeptical. I explained him the idea of chunks that would be called in other chunks. I explained him that this is just kind of macro expansion, not a function call. So, he answers: this is something that can be done with the preprocessor. I don't have arguments against this claim. What should I say?

From: **Lee Wittenberg**
Date: 31 Aug 1994

Denis Roegel writes: I was trying to convince somebody to use CWEB for his C programming. As usual, people are skeptical. I explained him the idea of chunks that would be called in other chunks. I explained him that this is just kind of macro expansion, not a function call. So, he answers: this is something that can be done with the preprocessor. I don't have arguments against this claim. What should I say ?

Unfortunately, you're fighting a losing battle. As a programmer who scoffed at literate programming until I had occasion to try it (is there anything worse than a converted sinner?), I know that human beings tend to be quite satisfied with the status quo, thank you, and resist learning new techniques as much as possible. "Although we've all been fed a load of psychological pap that says learning is fun an challenges are exciting, human beings are inherently conservative. Change means stress. Change makes us uncomfortable and fearful." (Michael Crichton, "Electronic Life", 1983)

As a CS teacher and literate programming proselytizer, I've found that a major problem is in perceptions. Most students ignore what I say about top-down design and stepwise refinement, because it seems as if the right way to do things takes too much time. No matter what I tell them (and they discover for themselves) about the time wasted debugging a poorly thought out program, they refuse to take the extra time up front. Literate programming has the same perception problem. Programmers look at the extra time necessary up front and don't see the (tremendous!) savings in debugging time. [sidebar: I recently hacked together a program to put Pascal type declarations in proper order. It was a fairly complex program, involving binary search trees, topological sorting, and the furshlugginer mess of lexical analysis in Pascal (even with the use of a non-standard string type). I finished the web in a week of afternoons, and was able to debug it in less than 2 hours, finding a total of 6 errors. If I had not been using literate programming techniques, it's arguable whether I could have completed the program in the same time, but I certainly couldn't have debugged it in as short a time. In addition, I'm fairly confident of the program's correctness as it stands, something that never happened before with a hacked-up tool before.]

However, if you wish to persist in your efforts to bring a new literate programmer into the fold, these are the advantages I believe that chunk names have over plain ol' macro definitions:

-- Typesetting of chunk names makes them easier to spot than macros, and encourages the use of descriptive phrases, which are understandable immediately, rather than short function-name-like abbreviations, which require at least one level of interpretation.

-- The descriptive nature of chunk names makes it easier to spot errors in the corresponding code. I once had a chunk name that (correctly) read @<Make sure |x| is greater than zero@>. The definition began with the typographical error ``if (x < 0)". I hate to admit how long I stared at the (non-working) code without seeing the obvious. I only discovered it by reading through the web, making sure that each definition agreed with its description.

-- Chunk names are cross-referenced by section number. Take a look at the code for CWEB. It's much easier to to follow the trail from a chunk name use to its definition, than for a macro (or function) call.

-- If you already have an algorithm, you can translate it into chunks in a single step (with a corresponding increase in confidence that the code will be correct). I contend that the following would be much messier using macro/function notation than chunks:

```
Algorithm  (from Bentley's "Programming Pearls", p. 36)

    initialize range to 1..N
    loop
        if range is empty,
            return that T is nowhere in the array
        compute M, the middle of the range
        use M as a probe to shrink the range
            if T is found during the shrinking process
            return its position

CWEB Translation
```

```
@d loop while(1)
@f loop else
@c
@<Initialize range to $1..N$@>;
loop {
    if (@<range is empty@>) {
        @<Return that |T| is nowhere in the array@>;
    }
    @<Compute |M|, the middle of the range@>;
    @<Use |M| as a probe to shrink the range;
         if |T| is found during the shrinking process,
         return its position@>;
}
```

Of course, Bentley translates the algorithm into more code-like form, but we can follow the same steps, with the added advantage that we don't have to replace the English placeholders with code ourselves -- CTANGLE does it for us. (Completing the translation is left as an exercise for the reader :-).

P.S. My debugging time for the Pascal program described above was actually less than one hour, but I didn't think anyone would believe that (and I'm still not sure I believe it).

From: **Jeffrey McArthur**
Date: 08 Aug 1994

I just had an interresting experience. I have been working 12-16 hour days, 7 days a week for the last month. Two days ago I realized I needed a utility. It had to process several files and create a new file as input for another application. I was really under pressure. The program was not trivial, but it was also not a major undertaking. The problem I had was two fold. First I needed it as soon as possible. Every hour it took to write was another hour at work and less time to finish before the stuff shipped. I started work on Saturday at 8:30 AM. I had to have the program finished, and the rest of the project complete by 9:00 PM on Sunday so it could be picked up by a courier to be taken to the client. (The utility would not be shipped to the client, it would just be used to create the output.)

Time was very tight. But there was also the problem that the logic required was a little convoluted. The final problem was I knew that this utility would be used many, many times in the future. So I threw convensional wisdom to the winds and decided to do it right the first time and write the utility using literate programming. This would mean I would write meaningful comments and documentation into a program written in very little time.

To my suprise, it took no extra time. There was a lot less debugging than I expected. The code did not run the first time, but the it only took a tiny amount of debugging to get it to work. I admit that some of the documentation is rough. I need to make sure it typesets properly and run it through a spelling checker. But it proved to me that it takes no extra time to use literate programming.

From: **Jeffrey McArthur**
Date: 12 Sep 1994

The following unsolicited quote (albeit from an literate programming convert) may prove useful for those of us who are trying to convince others that literate programming techniques really pay off in the long run: "I just had to pick up the stuff I wrote last November and revise it, and ... this was the first work I could be absolutely confident saying that if it wasn't a literate program, I would simply not have been able to revise it." (Carl Gregory, 12 September 1994) 'nuff said.

Converting Pascal-WEB programs into C-CWEB

From: **Andreas Scherer**
Date: 16 Sep 1994

Recently, I received the following question from Italy in a non-LP context: "Why doesn't Donald Knuth convert his TeX/MetaFont programs from WEB to CWEB?" Of course, I do know the answer(s) to this, but it lead me to the following: "Is there a simple tool for converting Pascal-WEB programs into C-CWEB sources, which keep the documentation around?" Can anybody answer this or provide such a tool?

From: **Marc van Leeuwen**
Date: 20 Sep 1994

Andreas Scherer writes: Recently, I received the following question from Italy in a non-LP context: "Why doesn't Donald Knuth convert his TeX/MetaFont programs from WEB to CWEB?" Of course, I do know the answer(s) to this, but it lead me to the following:

A slightly more pertinent question is of course why doesn't someone else do the conversion. I think it would be particularly useful if anyone thinks of making variants/extensions of TeX. I have heard a rumour that someone had hand-converted TeX to C and had thus obtained a very fast TeX on some particular platform. Unfortunately I have no precise indication of who or what; probably some of the features of high reliability and portability were sacrificed in the process.

"Is there a simple tool for converting Pascal-WEB programs into C-CWEB sources, which keep the documentation around?"

Not that I know of, since web2c operates on TANGLEd Pascal rather than directly on web, despite its name. It would be a good starting point for making a conversion program as suggested, though. Since web2c uses strict Pascal syntax in its yacc description, there are some problems to be solved. One is obviously to exclude the commentary fragments from the translation process (except pieces in `|...|', although as a first approximation you may wish to leave these as well), and to escape WEB-specific interjections (such as @^...@>) from the translation process; my first suggestion would be to replace the lex-generated lexical analyser by a hand-written one (I don't like lex anyway), probably best based on the original WEAVE, which already does much of this. Another problem is to make the syntax analyser flexible enough that it can recognise pieces of stuff without knowing their syntax category a-priori (just like WEAVE must); a possible advantage is that for module bodies only very few syntax categories will occur in practice. Two approaches are possible here: use yacc but add new categories that simply choose betwen a number of options without knowing beforehand (after all, yacc generates a bottom-up parser, so it should be possible to handle this to a certain extent), or to stick to WEAVE completely, and to modify its grammar in such a way that instead of TeX output it will produce CWEB output (a rather significant change of perspective). Both possibilities would make an interesting programming project, any voluteers? I would not expect that a completely automatic WEB->CWEB conversion will be feasible either way (web2c does not handle all of Pascal either), but a semi-automatic system for a significant subset of Pascal with some hand-editing of the result might be possible.

From: **Roland Kaufmann**
Date: 20 Sep 1994

Marc van Leeuwen and Andreas Scherer were discussing the how and why of converting Pascal WEB to CWEB code. Another way to do this conversion would be to run a Pascal to C converter on individual code chunks, leaving the TeX part of the WEB as it is (apart from web2c, there is Dave Gillespie's p2c, but I haven't tried it, let alone the significant amount of tinkering to get the web conversion to work).

What I am very sceptical about is what one could expect from such an automatic conversion, isn't it a contradiction to the tenets of literate programming? Who would like to read a work of literature passed through an automatic translation (I myself try to avoid even human translators when I can understand the original language)? While this comparison might be a bit far-fetched, I still believe that a web will require a substantial amount of editing performed by a reader with the necessary insight, computers aren't smart enough to do that.

If the reason for changing from Pascal to C is the (non)availability of the necessary compilers, the automated translation of the tangled code (as successfully demonstrated by TeX/METAFONT and web2c) might be all what's really necessary.

From: **Thomas Herter**
Date: 20 Sep 1994

Roland Kaufmann writes: Marc van Leeuwen and Andreas Scherer were discussing the how and why of converting Pascal WEB to CWEB code. Another way to do this conversion would be to run a Pascal to C converter on individual code chunks, leaving the TeX part of the WEB as it is (apart from web2c, there is Dave Gillespie's p2c, but I haven't tried it, let alone the significant amount of tinkering to get the web conversion to work).

You can not convert chunks of Pascal into C due to procedure nesting in Pascal, which is not present in C. For this reason (and some others) p2c need the _whole_ context information about the Pascal code to do the work.

Makefiles and configuration scripts in web files

From: **Dietrich Kappe**
Date: 24 Sep 1994

I've been using some simple perl scripts to convert old C and perl source into basic nuweb documents. Is anyone else automating the

conversion? With many programs I've found it easier to simply rewrite the things from scratch (not an option with 1000+ line IMHO). (On a related note...)

Also, whats the attitude on including makefiles and configuration scripts in the nuweb document. I myself prefer to keep all non source code elements outside of the nuweb doc. And how about manual pages? I've tried generating manual pages inside of the nuweb document, but documenting documentation makes my head hurt. I find it far more convenient using c2man and including comments before my function prototypes. Well, enough rambling., (Where are all the literati hiding? This used to be a high volume mailing.)

From: **Norman Ramsey**
Date: 25 Sep 1994

Dietrich Kappe write: Also, whats the attitude on including makefiles and configuration scripts in the nuweb document.

Value of this practice grows with size and complexity of makefiles and configuration stuff. Can also be valuable for distribution; sometimes you can ship a single file.

And how about manual pages? I've tried generating manual pages inside of the nuweb document, but documenting documentation makes my head hurt.

I have a new trick that seems to work well for this, but I'm not sure you can use it with nuweb. The trick is to put man pages, reference manuals, and the like in code chunks with special names (for example, names matching `refman:*'). The troff goes into these chunks, and I use "noweave -filter elide 'refman:*' " to make these chunks magically disappear when creating a source-code document. (The manual I just extract with notangle.) I'm not wildly ecstatic over this strategy, but it works much better than anything else I've tried.

From: **Barry Schwartz**
Date: 26 Sep 1994

Dietrich J. Kappe writes: Also, whats the attitude on including makefiles and configuration scripts in the nuweb document. I myself prefer to keep all non source code elements outside of the nuweb doc.

I have put the makefile into a noweb (not nuweb) source, but only for a very small program where the text needed fleshing out, and where the makefile had some literate merit of its own. (It generated an MS-DOS .com file rather than a .exe file, so someone might not figure that out if it were not there to read.) If a makefile is included, it should be as an example, rather than as a hard-and-fast-part-of-the-program, because makefiles are unstable. Configuration files, I believe, can benefit from being near the code to which they apply. This is especially the case if the configuration language is cryptic. Someone may hack with a configuration, though, so the configuration should be thought of as an example, rather than a necessary part (much as with makefiles). Test programs and test scripts should also be along with the code, or they are prone to get lost or go out of sync. I don't know about nuweb, but noweb lets you extract only those parts that you need at the moment, and you can use the cpif command to get some automatic control over selective extraction. So these ideas (humbly held) are easy to put into practice.

And how about manual pages? I've tried generating manual pages inside of the nuweb document, but documenting documentation makes my head hurt. I find it far more convenient using c2man and including comments before my function prototypes.

The noweb programs I've seen that included man pages all \iffalsed out the man pages from the printout. This is evidence that there is something subtly wrong with putting a man page in a noweb source. Not having the need to write man pages myself, I haven't dealt with this issue. I had thought about it, though, thinking more about Texinfo documents than nroff'd man pages. My opinion is that a Texinfo document is a piece of literature on the same level as a literate program, but with a different (though complementary) purpose, and so should be written as a separate document. I arrive at this view on literary grounds, rather than through arguments an illiterate programmer might make. There is a serious practical difficulty of keeping the Texinfo manual in sync, but that's life. There is also the possibility that the programmer may neglect the documentation of the code, saying that it will be taken care of in the Texinfo work. That, however, is exactly how program and manual may get out of sync. Ideally, there should be little information in the Texinfo document that is not also in the literate program, though the styles of writing may be very different. That way, you can verify that the Texinfo is correct, and also the program remains fully literate.

I'll make an exception for the case of a large program with an elaborate function. For instance, it is reasonable not to replicate the contents of "The TeXbook" inside "TeX: The Program". But suppose instead you are writing a grep program. Then I think the literate program should tell you how to run it; in other words, there should be instructions, probably near the beginning. The Texinfo manual, then, is a separate document, with a less terse set of instructions for running the program. I've only thought about these things, not practiced them. Except for the part about including instructions in the program. That's the approach, by the way, that the LaTeX 2e guys have taken. If you look at the tools packages, they all tell you how they are used, and then present the implementation.

(Where are all the literati hiding? This used to be a high volume mailing.)

Literate programming is so successful, that they are all busy writing reams of literate programs rather than up here begging for help.

From: **Joseph Brothers**
Date: 01 Oct 1994

I too find it practical to incorporate a Makefile in my noweb document for large, non-obvious or complex works (anything that takes a compile or tricky formatting for multiple file types or executing integral test scripts, or...). The Makefile can drive document preparation as well as software production. Using multiple targets, it can prepare different file types to be presented in the literate document's typeset output.

The example that follows shows how an applications' man page source, written in [[nroff -man]], can be made to appear alongside its readable version within the typeset book. It is excerpted from a 260-page book containing the client of a time recording application. In my opinion, side-by-side source and sample output of the man page is sufficiently explanatory to need no additional description. Note the noweb extension, "\include{tsmanpag.nw}". That's my own small contribution - sent to Norman Ramsey for possible future inclusion in noweb.

One gives the command "noweb tsclient.nw", then "make Makefile", then "make TimeSheet.n" to produce the man page, then "make tsclient.dvi" to view the typeset version onscreen, and "make hardcopy" to print 2-up on a Postscript printer. This is all discussed in the README, also incorporated in the document as a chunk. It's the "nroff -man TimeSheet.n | colcrt - >> tsmanpag.nw" that puts readable [[nroff -man]] output into the document. The [[{verbatim}]]'s preserve the work [[nroff]] did. Putting this pile of schlumpf in the Makefile keeps it handy and synchronizable. Putting the Makefile in [[noweb]] retains the possibility of understanding it three months from now.

```
_____________________________example_____________________________

\section{Make File}

Here's an excerpt from the complete project make file.

<<Makefile>>=
.nw.tex:
    noweave -n $? > $*.tex

tsclient.tex:   tsclient.nw tsusrifc.nw tsdbcomm.nw tsloclio.nw tswindow.nw tshlptxt.nw tsmanpag.nw
    ./noweb tsclient.nw

tsclient.dvi:  tsclient.tex
    latex $?
    noindex $?
    mv tsclient.nwi temp.nwi ; sort -u temp.nwi > tsclient.nwi ; rm temp.nwi
    latex $?
        xdvi tsclient

hardcopy: tsclient.dvi
    dvips -Pswi tsclient -o '!psnup -l -2 | lpr -Psw1'

TimeSheet.n:
    notangle -filter incp -RTimeSheet.n tsclient.nw > TimeSheet.n
    echo "\begin{verbatim}" > tsmanpag.nw
    nroff -man TimeSheet.n | colcrt - >> tsmanpag.nw
    echo "\end{verbatim}" >> tsmanpag.nw
@
\section{TimeSheet References}

The TimeSheet application is documented by a System Requirements
Specification and a man page. The TimeSheet man page is reproduced
below.

\section{TimeSheet Man Page}
\include{tsmanpag.nw}
\section{Man Page Source}
<<TimeSheet.n>>=
.TH TimeSheet 1 "8/12/94" " " " "
```

```
\"
\"
\"
.SH NAME
TimeSheet \- engineers' time tracking tool
.SH SYNOPSIS
.B TimeSheet

.SH DESCRIPTION
.PP
.I TimeSheet
is an online tool for recording and tracking a software engineer's time
by project and task. As of version 0.976, it runs only on Sun Sparcs
under the X windowing system. It will not run under SunView, nor on a
Sun 3.
@
```

From: **John Robinson**
Date: 01 Oct 1994

Makefile in CWEB file -- Whats the best way? The subject line says it all. Since CWEB has a tendency to remove tabs and to insert #line statements, it seems a bit difficult to specify Makefile definitions after "@(Makefile@>=" directives. Do any users have any suggests, or perhaps I should use another tool. I am usually programming in C++. Friends don't let friends do DOS.

From: **Lee Wittenberg**
Date: 02 Oct 1994

John Robinson writes: Makefile in CWEB file -- Whats the best way? The subject line says it all. Since CWEB has a tendency to remove tabs and to insert #line statements, it seems a bit difficult to specify Makefile definitions after "@(Makefile@>=" directives. Do any users have any suggests, or perhaps I should use another tool. I am usually programming in C++.

You can always use @= to insert verbatim text in a code chunk. This is probably a good idea, since CWEB will do strange typesetting things to your makefile stuff. You should also redefine \vb=\. to get rid of the boxes woven around verbatim code. You can also use noweb, nuweb, etc.

P.S. I've generally refrained from including makefile information in my webs. The chicken & egg problem of "how do you make the makefile" bothers me philosophically. Then again, CWEB will always create a new makefile whenever you make the .c (or .cc) file. This could confuse the operating system. I generally tend to think of a makefile as a metalevel in the process, so it doesn't actually belong in the web (or is that just a rationalization?).

From: **Barry Schwartz**
Date: 03 Oct 1994

John Robinson writes: Makefile in CWEB file -- Whats the best way? The subject line says it all. Since CWEB has a tendency to remove tabs and to insert #line statements, it seems a bit difficult to specify Makefile definitions after "@(Makefile@>=" directives. Do any users have any suggests, or perhaps I should use another tool. I am usually programming in C++. Friends don't let friends do DOS.

And friends don't let friends put makefiles in CWEB webs. I think your problem is that you are using a plier where a wrench is needed. If you want to mix languages, use a tool that lets you mix languages. I like noweb. I think CWEB imposes too much composition style, anyway.

From: **Joseph Brothers**
Date: 04 Oct 1994

I prefer noweb becuase it mostly stays out of my way. The 2.6c version of noweb knows enough about C to meet my minimum needs, including some form of pretty-printing (via the embedded makefile and \include{}) and cross-referencing. Here's a scrap from my noweb document. It builds the document's Makefile. The Makefile not only weaves the document, it tangles the software source and compiles and installs it too. Note the 'unexpand' command, a handy Unix tool necessary for proper "tabination" of the output. The incp filter is something I built to permit noweb documents to include other noweb documents.

```
\section{MakeFile}
```

```
<<Makefile>>=
Makefile: tsclient.nw
    notangle -filter incp -RMakefile tsclient.nw | unexpand > Makefile

@
```

Cross-referencing in CWEB

From: **Charles Blair**
Date: 03 Oct 1994

Is there a simple way to refer to a later module in the text part of a module. Something like this:

```
@
The lists are sorted in @<sorting the lists@>.
@c
    ....

(much later)

@
@=
...
```

with CWEB supplying the appropriate module number and perhaps expanding an abbreviation for the module name.

From: **Denis Roegel**
Date: 04 Oct 1994

Try

```
@
The lists are sorted in |@<sorting the lists@>|.
@c

 ....
```

From: **Marc van Leeuwen**
Date: 04 Oct 1994

Yes there is, just write

```
@
The lists are sorted in |@<sorting the lists@>|.
@c
```

i.e., put the module name in C mode. Doing this or making forward references to abbreviated module names are no problem in Levy/Knuth CWEB 3.0 or newer. [These things will also be possible in the new release of my implementation of CWEB that was (foolishly) called CWEB 3.x previously, but will henceforth be named CWEBx, but that is besides your point. What may be interesting is that that version allows you to refer to a section number only, by writing

```
@
The lists are sorted in section @#list sorting section@>.
@c

(much later)

@ @:list sorting section@>
Here we sort the lists
```

I find this more useful because (1) it also works if the lists are sorted in an unnamed section (one with @c) or (less likely) in the

continuation of a module that is defined in multiple sections, (2) it is also possible to refer to pure-documentation sections, (3) the phrase "The lists are sorted in |@<sorting the lists@>|" seems to have some redundancy, which can get rather tiring if the module name is really long.

From: **Lee Wittenberg**
Date: 05 Oct 1994

Yes. In CWEB 3.0 (& up), you can say

```
@
The lists are sorted in |@|.
@c
...
```

which works just as you would expect.

From: **Barry Schwartz**
Date: 05 Oct 1994

Charles Blair writes: Is there a simple way to refer to a later module in the text part of a module.

Instead of depending on CWEB to do all the text processing for you (it can never be powerful enough, and isn't very extensible), you can try using Karl Berry's eplain macros. Then you can set up any sort of cross-links scheme you want, using the \ref mechanism. With eplain, as with LaTeX, you have to do multiple passes over the document to resolve these things. You get a lot of other stuff, besides. For instance, there are macros for drawing commutative diagrams. I haven't drawn too many of those lately, but I was able to adapt eplain to draw pictorials of skip lists, within a CWEB program.

Wishes for CWEB features

From: **Yotam Medini**
Date: 04 Oct 1994

Let me skip the standard excuses of being a new CWEB baby-user. I strongly appreciate CWEB core ideas, but I believe that CWEB will never gain the wide acceptance it deserves as long as its output is frequently worse than what the typical programmer expects. Here are the features that I wish would be implemented in CWEB.

CTANGLE: * Beautify and open the control of the output. This is important when using CWEB to produce exported header files that may be used by non-CWEB/TeX-speaking programmers.

CWEAVE: * In DEK's "The Stanford GraphBase" there are wonderful mini-indexes on the odd right-side pages. In page 576 the paragraph titled 'About this book' reveals that these mini-indexes were produced by CTWILL. The source for CTWILL is located in: labrea.stanford.edu:/pub/ctwill. Unfortunately, it seems that it is not in a 'production-reliable-status'. Quoting from ctwill.w: "\.{CTWILL} was hacked together hastily in June, 1992, to generate pages for Knuth's book about the Stanford GraphBase, and updated even more hastily in March, 1993 to generate final copy for that book. The main idea was to extend \.{CWEAVE} so that "mini-indexes" could appear. No time was available to make \.{CTWILL} into a refined or complete system, nor even to fully update the program documentation below." Let us have it as a cweave full-working option! * This is "religious-related". Open control to styles of where to put braces, and when to insert new-lines. * Fix nesting losses. Frequently, after some long 'for' loop header or some several nested loops, cweave gives up on push-right-indenting the code. For example try cweave+tex+xdvi the following:

```
------------------------------ nest.w ------------------------------
@* WEB-nesting.
Try this:

@c
static int
filter_single_non_addressed(BLOCK *pb)
{
   static const char *err_colon_fmt = "colon not found";
   int   n_rec, n_names, n_single, n_output, n_curr_name /* statistics */;
```

```
  for (n_rec = n_single = n_output = n_names = n_curr_name = li = 0, lp =1;
       fgets(line[li], sizeof(line[0]), pb->fdump);
       n_rec++, lp = li, li = 1 - li)
  {
     pcolon1[li] = strchr(line[li], ':');
     if (pcolon1 == NULL)
     {
        error_exit(pb, err_colon_fmt, pb->tmp_fn, n_rec+1, line);
     }
     *pcolon1[li] = '\0'; /* trim line for name */
     if (strcmp(line[0], line[1]))
     {
        /* a new name */
        n_names++;
        n_curr_name = 1;
     }
     else
     {
        n_curr_name++;
     }
  }

  return(1);
}
```

---

*Alignment control. Let me quote from X11/Xlib.h:

---------- Xlib.h ---------- typedef struct { XExtData *ext_data; /* hook for extension to hang data */ Font fid; /* Font id for this font */ unsigned direction; /* hint about direction the font is painted */ unsigned min_char_or_byte2;/* first character */ unsigned max_char_or_byte2;/* last character */ unsigned min_byte1; /* first row that exists */ unsigned max_byte1; /* last row that exists */ Bool all_chars_exist;/* flag if all characters have non-zero size*/ unsigned default_char; /* char to print for undefined character */ int n_properties; /* how many properties there are */ XFontProp *properties; /* pointer to array of additional properties*/ XCharStruct min_bounds; /* minimum bounds over all existing char*/ XCharStruct max_bounds; /* maximum bounds over all existing char*/ XCharStruct *per_char; /* first_char to last_char information */ int ascent; /* log. extent above baseline for spacing */ int descent; /* log. descent below baseline for spacing */ } XFontStruct; ----------

It is obvious that the code above is meant to be aligned for both the field name and the field comment. I could think of two approaches for letting the final TeX'ed output to look nicely aligned. 1) Introduce a new control ('@a') sequence, where the user will be able to define 'tabulator points' on the fly. (@a on a single line by itself will reset). For example:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
typedef struct {
    @a          @a                  @a
    XExtData    *ext_data;          /* hook for extension to hang data */
    Font        fid;                /* Font id for this font */
    unsigned    min_char_or_byte2; /* first character */
    int         descent;            /* log. descent below baseline for spacing */
@a
} AlignedStruct;
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

2) Make cweave smart, to identify such aligned blocks in the source. Such sensitivity could be controlled by some parameters such as: how many spaces before a text it takes to make it a candidate for an alignment, and how many consecutive source lines with similar alignment should tell cweave that this is not a coincidence.

From: **Timothy Larkin**
Date: 04 Oct 1994

Yotam Medini writes: Fix nesting losses. Frequently, after some long 'for' loop header or some several nested loops, cweave gives up on push-right-indenting the code. For example try cweave+tex+xdvi the following: @<example deleted@>

Actually, CWeave formats this perfectly well. However, you need to be aware that you can confuse CWeave by failing to keep it informed about the meanings of symbols. I most frequently encounter this myself when I use a Mac ToolBox type that has no definition in my web source. Your example makes the twin error. CWeave thinks that "line" is the line directive, whereas you use it as

a variable name. If you include the instruction, @f line x, before @c, to tell CWeave that "line" should be treated as a simple variable, CWeave will format the code correctly.

@<example struct definition deleted@> It is obvious that the code above is meant to be aligned for both the field name and the field comment.

Your suggest that the programmer have tab stops to play around with has some merit. However, I prefer to leave all these decisions to CWeave. This is the advantage (and disadvantage) of automatic pretty printing.

From: **Marc van Leeuwen**
Date: 05 Oct 1994

Yotam Medini writes: I strongly appreciate CWEB core ideas, but I believe that CWEB will never gain the wide acceptance it deserves as long as its output is frequently worse than what the typical programmer expects. Here are the features that I wish would be implemented in CWEB.

Some of the features you mention are implemented in my implementation CWEBx of CWEB (previously called CWEB 3.x) that is now in beta testing. I find that in this version I can get almost all output to look as I wish if a bit of attention is paid to it; partly this is because I was in a position to adapt the grammar to my wishes, but all the style variation I wanted are implemented as command line options, so you can still get the (ugly:-) style variations that other people seem to prefer just as easily. References to CTANGLE and CWEAVE below are to the programs of CWEBx.

CTANGLE: Beautify and open the control of the output. This is important when using CWEB to produce exported header files that may be used by non-CWEB/TeX-speaking programmers.

CTANGLE (CWEBx) has a `-l' switch that will turn off #line directives, preserve spacing and comments from the input, and even tries to construct a proper indentation for inserted uses of modules.

** In DEK's "The Stanford GraphBase" there are wonderful mini-indexes on the odd right-side pages. In page 576 the paragraph titled 'About this book' reveals that these mini-indexes were produced by CTWILL. The source for CTWILL is located in: labrea.stanford.edu:/pub/ctwill Unfortunately, it seems that it is not in a 'production-reliable-status'.*

Remarkable. I thought the mini-indexes were a kind of private secret that Knuth kept to ensure that some copyright could be attached to the book of which the sources are almost completly public (see the copyright notice that explicitly mentions special indexes). Aparently I was mistaken. It seems promising to try to include this in the standard CWEAVE program, but it will take a bit of work.

** This is "religious-related". Open control to styles of where to put braces, and when to insert new-lines.*

As mentioned above, this can be done in my CWEAVE. The options are: +u: unaligned braces (`{' dangling at end of line) as in Levy/Knuth CWEB. +w: wide braces (`{' on a line of its own) (default) (`{' aligned with `}' at beginning of a line, but followed (usually) by a statement indented one stop, aligning with the next line. +f: force statements on a new line. +a: even more so: even if statement follows `if(..)' or a label. +m: do not force declarations in a local block on separate lines.

** Alignment control. [example deleted] It is obvious that the code above is meant to be aligned for both the field name and the field comment. I could think of two approaches for letting the final TeX'ed output to look nicely aligned. 1) Introduce a new control ('@a') sequence, where the user will be able to define 'tabulator points' on the fly. (@a on a single line by itself will reset). For example:*

```
|> - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
|> typedef struct {
|>     @a          @a                @a
|>     XExtData *ext_data;         /* hook for extension to hang data */
|>     Font        fid;              /* Font id for this font */
|>     unsigned min_char_or_byte2; /* first character */
|>     int              descent;          /* log. descent below baseline for spacing */
|> @a
|> } AlignedStruct;
|> - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Two difficulties: CWEAVE is normally not very aware of the column of the input line it is on (tabs are weird things, and not everybody likes them to be 8 spaces wide), and the output has to be breakable accross lines if they get to wide for the page, which is not easy to achieve in a tabular context.

2) Make cweave smart, to identify such aligned blocks in the source. Such sensitivity could be controlled by some parameters such as: how many spaces before a text it takes to make it a candidate for an alignment, and how many consecutive source lines with similar alignment should tell cweave that this is not a coincidence.

This is even harder to define or to implement. Care to try?

From: **Barry Schwartz**
Date: 05 Oct 1994

Yotam Medini writes: Let me skip the standard excuses of being a new CWEB baby-user. I strongly appreciate CWEB core ideas, but I believe that CWEB will never gain the wide acceptance it deserves as long as its output is frequently worse than what the typical programmer expects. Here are the features that I wish would be implemented in CWEB.

With CWEB, there'll always be something someone wants and can't have, because it tries to do too much in its kernel. Witness the recent complaint in this newsgroup from someone trying to put makefiles in the CWEB sources. CWEAVE tries to be the perfect C programming tool right out of the box, and so comes up completely unable to handle makefiles. And if you are going to get all the C formatting choices that you want, it will all have to be hacked into CWEAVE, and then every CWEB user will have to shlep it around whether he or she uses it or not. And then CWEB will be a moving target, like the shrink-wrap we've all come to know and hate.

Furthermore, CWEB will never take off because C is in decline. I believe CWEB is too difficult to use with C++. And what good's this language dependence, anyway? I'd rather write a program in Icon than in C, if I can get away with it. Why should I have to use C to be literate? Why can't I make my Perl scripts and stuff-written-in-the- special-purpose-language-developed-just-for-this-project literate?

I think that, if there is a future in literate programming (and I think there is), that it looks more like noweb than CWEB. Noweb takes advantage of the development environment, using it to make weaving and tangling flexible and extensible. You lose the near-complete platform independence of CWEB, but noweb is so simple that it can be ported to a new operating system in a short while, if the OS is powerful enough and supports C (and C++, if you use Lee Wittenberg's MS-DOS port of noweb as your base) and Icon (which is portable and free). And noweb is extremely simple to use.

What else do you lose? Well, there's no built-in prettyprinting, though it can be added as a filter if you want to go to the trouble. Prettyprinters already exist for a few languages (C not included). But program text in monospaced font is adequate, I have found, and the human-language writing comes out much better than with CWEB, because I can use a standard article format. (I haven't tried writing a literate program as a letter, but I have the idea in the back of my mind.) Automatic indexing is not so fancy as with CWEB, but tools to handle that are improving--as I said, noweb is extensible. And you can manually index any language at all. If you want, you can use standard LaTeX indexing tools to get a traditional index, as well. You can add a bibliography, or anything else like that. I've been able to do a bibliography with CWEB, by using eplain macros, but it's always an add-on with CWEB. With noweb, it's standard LaTeX.

And you don't have to use LaTeX. You can generate output for plain TeX, or HTML, or LaTeX2HTML. Or anything else, if you care to write the back-end (for which you don't have to rewrite all the other components). Did I mention that noweb is almost trivial to use?

From: **Preston Briggs**
Date: 06 Oct 1994

Yotam Medini writes: Here are the features that I wish would be implemented in CWEB.

"If you want something done right, ..." That is, nobody's going to do these things for you. However, if you're willing to explore a bit, other people have come up with some alternatives to CWEB.

** Beautify and open the control of the output. [of tangle] * In DEK's "The Stanford GraphBase" there are wonderful mini-indexes. * This is "religious-related". Open control to styles. of where to put braces, and when to insert new-lines. * Fix nesting losses. 1) Introduce a new control ('@a') sequence, where the user will be able to define 'tabulator points' on the fly. 2) Make cweave smart, to identify such aligned blocks in the source.*

Check out noweb, which solves all your problems and more, or nuweb which gets everyhting but the mini-indices (because I could never get them to look pretty enough). Of course, you may not like the "solutions" in some cases! A lot of this is a matter of personal taste, experimentation, and compromise.

From: **Tim Kientzle**
Date: 07 Oct 1994

Marc van Leeuwen writes: Some of the features you mention are implemented in my implementation CWEBx of CWEB (previously called CWEB 3.x) that is now in beta testing...

In merging CWEAVE output into LaTeX documents, I've run across one other fairly major stumbling block, which is that CWEAVE inserts _visual_ formatting rather than _logical_ formatting. In the work I'm doing, it would be very nice to have identifier definitions marked-up differently from indentifier uses, so that identifier definitions could be automatically indexed by LaTeX, for example. I've looked through CWEAVE, and this looks doable, but I've not yet tried to modify CWEAVE in this manner. This sort of logical markup would allow the use of standard LaTeX packages to compile multiple indices (e.g., an identifier index at the end of each section, yet all function definitions are also indexed at the end of the book), and other sorts of cross-referencing tricks. If you add this sort of facility to CWEBx, I'm definitely interested.

P.S. I've heard a lot about noweb, which sounds like it could be much more appropriate for certain purposes. However, I'm using CWEB to write chapters for books, and really appreciate the pretty-printing and automatic indexing and cross-referencing.

From: **Marc van Leeuwen**
Date: 07 Oct 1994

Tim Kientzle writes: In merging CWEAVE output into LaTeX documents, I've run across one other fairly major stumbling block, which is that CWEAVE inserts _visual_ formatting rather than _logical_ formatting. In the work I'm doing, it would be very nice to have identifier definitions marked-up differently from identifier uses, so that identifier definitions could be automatically indexed by LaTeX, for example. I've looked through CWEAVE, and this looks doable, but I've not yet tried to modify CWEAVE in this manner. This sort of logical markup would allow the use of standard LaTeX packages to compile multiple indices (e.g., an identifier index at the end of each section, yet all function definitions are also indexed at the end of the book), and other sorts of cross-referencing tricks.

I sympathise with your wishes, but I don't quite understand your first sentence. Of course CWEAVE does visual formatting, at least for the part that cannot be left to be visually formatted by TeX: it issues forced line breaks, explicit changes of indentation level etc. The logical markup is really contained in the source document, and not (fully) in the intermediate file produced by CWEAVE. However CWEAVE does not do complete visual formatting: for instance many symbols like operators are produced as macros, so that you can easily change their appearance by overriding the default definitions, and breaking of long lines is also left to TeX's line breaking algorithm.

However, I think this is not really what bothers you. You want CWEAVE to export some of the information about the program that it has access to while processing, although it is not represented explicitly in the input: whether a certain occurrence of an identifier is defining or using. This is analogous to other cases where CWEAVE does make such information explicit, for instance it marks unary and binary uses of operators differently, making the output from |**p| different from $**p$ (where TeX would make the second asterisk a binary operator). And you are right, definitions and uses of identifiers generally trigger different syntax rules, so that CWEAVE could easily make the distinction; in fact it does, but currently this only influences the underlining in the index[*].

To make the change you wish one would have to change the internal coding for identifiers, which currently allows only two attributes---for normal identifiers and reserved words---so as to allow one or two more distinguished variants (defining identifiers and maybe defining reserved words (such as typedef symbols)), which could be altered by |make_underlined| and could determine which macro to use upon output. You have to be careful not to exceed the sixteen bits used to represent identifiers currently, but apart from this it would not be a very difficult change to make. If you want to distinguish really many uses of identifiers (function definitions, local variables, macros, ...) you may want to change the internal coding of identifiers in a more fundamental way.

If you add this sort of facility to CWEBx, I'm definitely interested.

Well, I'm not planning to do this on the short term, particularly because within the framework for which CWEB is intended (namely for producing literate presentations of program sources in a fairly standardized format, rather than producing more or less arbitrary documents that happen to contain pieces of program code), it is not quite clear which purpose it would serve, and therefore exactly which kinds of information should be exported. But anybody is invited to make the changes of his liking; CWEBx was (re)written to be more easily understandable and less rigid than CWEB (even though I think it could still bear some improvement in this respect).

```
[*] Note however that the logic is not infallible, due to a combination of
    the brain-damaged syntax of C/C++ for declarations and the weird grammar
    used by CWEAVE, making that in

    char *p;
    char (*q);
```

```
long (*f) (void);
unsigned short (*g)(void);
const int *(*h)(void);
char *(* volatile *k)(void);

the identifiers |p|, |short|, |h|, and |volatile| are considered to be
defining, and none from the second and third lines; here at least CWEBx
does a more decent job, tripping only over the final case).
```

From: **Norman Ramsey**
Date: 07 Oct 1994

Tim Kientzle writes: P.S. I've heard a lot about noweb, which sounds like it could be much more appropriate for certain purposes. However, I'm using CWEB to write chapters for books, and really appreciate the pretty-printing and automatic indexing and cross-referencing.

noweb does automatic indexing and cross-referencing for some languages, including C, Icon, TeX, and Standard ML. Some people have made it prettyprint other language, like Icon and Object-Oriented Turing. I haven't been overwhelmed by the results, but then I'm notoriously difficult to convince of the value of prettyprinting.

From: **Barry Schwartz**
Date: 08 Oct 1994

Norman Ramsey writes: noweb does automatic indexing and cross-referencing for some languages, including C, Icon, TeX, and Standard ML. Some people have made it prettyprint other language, like Icon and Object-Oriented Turing. I haven't been overwhelmed by the results, but then I'm notoriously difficult to convince of the value of prettyprinting.

I used to think that prettyprinting was the cats meow, but after becoming a bit accustomed to noweb I find that a CWEB printout looks somewhat like gibberish.

I think the problem here is not so much that prettyprinting is inherently bad as that many languages don't benefit from it so much. Algol and Pascal, I believe, benefit quite a bit from appropriate prettyprinting. These are languages where you write out a lot of whole words: "begin", "end", "do", "then".... You get the idea. Languages like C and Icon depend a lot on symbolic notation. For the large part, all CWEB does is replace one set of symbols with another that some people happen to prefer. But, still, CWEB puts reserved words and defining words in boldface and identifiers in italics; what of that? In Pascal, doing that helps mark out the form of a construct. In C, on the other hand, the constructs are delimited by things like parentheses and braces, and so putting things in different typefaces doesn't help that much. OK, CWEB indentifies, but you don't need weave to do that; Emacs can do that for you, or you can do it yourself, and the web source will benefit from it. The main thing that I might want from noweb but don't get is typesetting of comments, and I can live without that because I don't need a great many comments.

Not-small C program with noweb

From: **Eric Prestemon**
Date: 12 Oct 1994

I am making my first foray into writing a more-than-trivial C program and my first foray into noweb. I have a few questions. Are there examples of finished C/noweb programs out there, beyond what is in the noweb distribution? Specifically, I am interested in how people write functions and function calls cleanly (rather than inlining a chunk, which seems the more natural *WEB way to do things).

Is noweb less usable than, say, CWEB or FWEB for this kind of thing? The printed output isn't too critical to me; only I and whoever maintains the program after I stop working on it will ever see it. I don't plan to spend much (any) time customizing the output. Also, a DOS implementation is handy, though much of my development is on a unix machine. For these reasons, and my plan to use multiple output files, I think noweb is best, but I'm open to other ideas. (As an aside, I'm not much of DOS guru, and the dosnoweb-2.5a distribution didn't include a cpif, as far as I can tell. Is there an easy way to do such a thing in DOS? Between having multiple output files and a slow PC, only compiling the source files that change would save me a lot of time.)

Is there a good emacs mode for editing noweb files? I've been using C mode, which I guess is okay, but it's not perfect. Diving into the cc-mode.el code to make changes seems too difficult. This isn't critical, cc-mode gets the C parts right, and I just have to convince it to let me enter certain things in the text. Thanks for any help or any other comments,

---

From: **Norman Ramsey**
Date: 13 Oct 1994

Eric Prestemon writes: Are there examples of finished C/noweb programs out there, beyond what is in the noweb distribution?

This is a good question. A while back (last year?) there was some idle talk about setting up an archive of literate programs for people to read. This would let people read some literate programs and compare tools and techniqus. George Greenwade even said he might be able to provide space for such an archive after his new disk arrived. This is a task worth doing, but it would take nontrivial work to set up and maintain such an archive. (For example, do you keep source? .tex? postscript? all of the above.) Any volunteers?

To answer your specific question, I don't know of any well-written C programs using noweb that are readily available. The best-written noweb program I have seen is about to be published as a book, but you can't get it yet:

```
  @book{fraser:retargetable:book,
    author="Christopher W. Fraser and David R. Hanson",
    title="A Retargetable {C} Compiler: Design and Implementation",
    publisher="Benjamin/Cummings", address="Redwood City, CA",
    year=1995, note="ISBN 0-8053-1670-1, in press"
  }
```

Don Knuth's latest book, The Stanford GraphBase, contains a number of CWEB programs that are worth some study. If you're learning C and literate programming, reading CWEB and writing noweb should present you no problems (writing CWEB is another story). Marcus Speh has a few examples on the WWW; try http://info.desy.de/user/projects/LitProg.html.

Is noweb less usable than, say, CWEB or FWEB for this kind of thing?

I claim noweb is *more* usable than CWEB or FWEB. See my articles in the September 1994 IEEE Software or the July 1991 Software---Practice & Experience.

Is there a good emacs mode for editing noweb files?

There are some emacs modes, including one distributed with noweb, but I think the world is still waiting for a good one...

From: **Lee Wittenberg**
Date: 13 Oct 1994

*Eric Prestemon writes: Are there examples of finished C/noweb programs out there, beyond what is in the noweb distribution? Specifically, I am interested in how people write functions and function calls cleanly (rather than inlining a chunk, which seems the more natural *WEB way to do things).*

I don't know of any, unless you count the C++ programs in the DOS distribution. One of these days I hope to get some more samples to pub/leew/samples.LP at bart.kean.edu, but as of yet, all the C examples there use CWEB. The technique I generally use for C programs is an overall structure something like (I use this in CWEB, too):

```
<<[[#include]] lines>>
<<Global variables & constants>>
<<Function prototypes>>
<<Function definitions>>

Then, whenever I define a function, I write 2 chunks. For example:

<<Function prototypes>>=
int func(int, int);
<<Function definitions>>=
int func(int x, int y)
{
        return x + y;   /* what, you want a *real* function? */
}
```

I find this technique works very well, particularly since the prototype and the function definition stay in close proximity in the web. If one changes, the appropriate change is easy to make to the other.

Is noweb less usable than, say, CWEB or FWEB for this kind of thing? The printed output isn't too critical to me; only I and whoever maintains the program after I stop working on it will ever see it. I don't plan to spend much (any) time customizing the output. Also, a DOS implementation is handy, though much of my development is on a unix machine. For these reasons, and my plan to use multiple output files, I think noweb is best, but I'm open to other ideas.

In my experience, they are all equally usable. They each have their strengths and weaknesses. You pays your money and you takes your choice. noweb is probably the easiest of the three to learn, however.

(As an aside, I'm not much of DOS guru, and the dosnoweb-2.5a distribution didn't include a cpif, as far as I can tell. Is there an easy way to do such a thing in DOS? Between having multiple output files and a slow PC, only compiling the source files that change would save me a lot of time.)

Barry Schwartz has written one for DOS, it just hasn't made it into the official DOS package yet (mea culpa). By the way, the current DOS package supports version 2.6c, and I recommend the upgrade.

From: **Barry Schwartz**
Date: 14 Oct 1994

*Eric Prestemon writes: Are there examples of finished C/noweb programs out there, beyond what is in the noweb distribution? Specifically, I am interested in how people write functions and function calls cleanly (rather than inlining a chunk, which seems the more natural *WEB way to do things).*

I haven't yet written any large programs with noweb, but even in small programs I don't depend too much on the chunk mechanism, unless the language makes subroutines ungainly. (Perl may exhibit this property.) I like to put C subprograms in two main chunks. First I present a prototype, and then the definition, adjacent to each other. The prototype goes into a chunk that gets tangled to near the top of the C file, and (in a multiple-file program) to a header file. The definition can go almost anywhere in the C file.

Some languages are much easier to handle. In Icon, for instance, globals can go anywhere, and there's no need for prototypes or the like. This works out very well; you can just put all globals in the root chunk, or wherever you like.

(As an aside, I'm not much of DOS guru, and the dosnoweb-2.5a distribution didn't include a cpif, as far as I can tell.

I have written a cpif in C/noweb, for MS-DOS, but it hasn't been released yet. However, there is nothing outlandish about the code; all I did was duplicate the functions of the shell script.

Empty space in noweb TeX output

From: **Joerg Viola**
Date: 15 Oct 1994

A question concerning the documentation output formatting of noweb: It appears to me that sometimes, at the start of a documentation chunk, a new page is started, even if the old one is not nearly full. This results in pages containing only some lines of docs and some of code and 80 percent emptiness. What to do?

From: **Norman Ramsey**
Date: 16 Oct 1994

This ugly behavior is the result of an unpleasant compromise. I decided that at all costs, code chunks should fit on one page. Therefore, there is a \filbreak in the definitions of \nwendcode and \nwdocspar. Changing these definitions may give results you like better, at the cost of breaking code chunks across pages. See tex/support.nw in the noweb distribution for more info.

Style of header file inclusion

From: **Felix Gaertner**
Date: 17 Oct 1994

I'm currently going over some code I wrote a year ago and came across an interesting point concerning literate programming style. I am interested in how people write the inclusion of header files `literally'? (I am using CWEB, but that's not so important for the style.) A year ago I used to do this:

```
------- begin example 1----------------------------------------
@ This is the overview over the program.
@c
@<What headers do we need?@>
@<Functions@>
@<The main program@>

[later:]

@ We've used the |foo| function so we need to include its header.

@<What headers do we need?@>=
#include"foo.h"
...
------ end example 1 ----------------------------------------
```

During the course of the program all #includes were added to this one module when it seemed suitable. But on the other hand, I also used this style, the point being that common headers were mentioned directly in the unnamed module, even if their use wasn't directly visible already.

```
------- begin example 2 ----------------------------------------
@ This is an overview...
@c
#include<stdio.h>
#include<string.h>
@<Other header files to include@>
@<Functions@>
@<The main program@>
...
-------- end example 2 ----------------------------------------
```

I wonder how other people do this? It would also be interesting to know which names people give to the module that `collects' all the #includes? (e.g. <What headers do we need?>, <Header files to include>, etc.) I wonder if I have made my point clear? It's a simple stylistic issue, I believe.

From: **Roland Kaufmann**
Date: 18 Oct 1994

In the hope that this is not too lengthy, some excerpts from a multi-file CWEB I have written some time ago. It was converted to "literate" form after writing in pure C (someone else wrote the code, I tried to make easier to understand for myself and hopefully others. Note that I use colons to indicated omitted parts of the file and meta-comments added for this posting starting with a double percent sign.

```
-- drs-pp.w ----------------------------------------------------------
        :
        :
@ The programme layout.

Literate programming allows to present the code in small pieces which
makes it much easier to get an overview of a complex piece of
software. The programme \.{\ProgramName} consists of several \Cee\ files:

\medskip
\halign{\indent#\hfil&\qquad#\hfil\cr
  \.{drs-pp.c}  & |main()| and some auxiliary routines.\cr
  \.{config.h}  & Machine and operating system dependencies.\cr
```

```
  \.{drs-pp.h}  & Declaration of common objects.\cr
  \.{btb.c}     & Handling of BOSS standard tabular
                  files~[COM91,~section~I].\cr
  \.{memory.c}  & Handling of multi-dimensional arrays.\cr
  \.{cubspl.c}  & Cubic spline interpolation.\cr
  \.{process.c} & Computation of degradation and two-dimensional
                  cuts.\cr
  \.{phnoise.c} & Computation of error rate with phase noise.\cr
  \.{math.c}    & Numerical integration, special functions.\cr
  \.{mdfile.c}  & Access to BOSS multi-dimensional files.\cr
}
% This has to be updated MANUALLY!
\medskip

which are produced from the corresponding \.{CWEB} sources \.{*.w}.
The header files are treated differently---the declarations in
\.{drs-pp.h} originate in the \.{CWEB} source where the objects
(functions, variables) are defined. The ``master'' file \.{drs-pp.w}
ensures that all the declarations are tangled into the file
\.{drs-pp.h} where each \.{*.c} file can include it to allow separate
compilation.

On the other hand, the file \.{config.h} is standard \Cee\ and
contains \Cee-preprocessor constants to adapt the programme to
different environments.

@ The file \.{drs-pp.c} contains the |main()| function and a few
auxiliaries. In contrast to all other \Cee\ files, which use the
``file module'' construct of CWEB~[Lev90], this one will be tangled
straight from the following ``unnamed module''. A warning message is
inserted in a \Cee\ comment at the beginning of each file.
Modifications should be done in the \.{CWEB} source or using the
change file mechanism~[Lev90,Knu84,Knu89a] and not in the \Cee\ code!
% the AT/ forces a line break
% This is the only ``unnamed module''!

@c
@=/* This file (drs-pp.c) has been produced automatically from 'drs-pp.w'.@>@#
@=   You almost certainly shouldn't edit it! */@>@#
  @<Local |#include|\thinspace s@> @;
  @<Local and global |static| variables @>@;

  @<Print error message: |error()|@> @;
  @<Read one line: |fgets_n()| @>@;
  @<|screen_output_main()| @>@;
  @<|screen_output_cut2d_1()| @>@;
  @<|screen_output_cut2d_2()| @>@;

  @<The |main()| function@> @;

@ We need standard input and output, string functions and character
classification from the \Cee\ library. Everything common to all \Cee\
files should be in \.{drs-pp.h}.

@f const static

@<Local |#include|\thinspace s@>=
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include "drs-pp.h"

@ Global |#define|\thinspace s. As we have several \Cee\ files, we need to
distribute definitions which should be visible in all modules via a
header file which is |#include|\thinspace d by each \Cee\ file.

@(drs-pp.h@>=
@=/* This file (drs-pp.h) has been produced automatically from 'drs-pp.w'.@>@#
```

```
@=   You almost certainly shouldn't edit it! */@>@#
#include "config.h"
@<Define global constants@>@;
@<Declare global variables @>@#
        :
@<Declare functions for |cubspl.c| @>@;
        :
%% Larger pieces are included from separate webs like this:

@* Cubic Spline interpolation.

@i cubspl.w
        :
%% This is added to make this sort of self-consistent
@* References.

% For this short (?) document, ``hand made'' references are used---I
% simply do not know how to make automatic references (like LaTeX's
% \cite) in plain TeX. This is from the TeXbook, page 340 adapted to
% my personal stylistic preferences.

\medskip
{\frenchspacing                 % put everything in a group to avoid
                                % side effects

\item{[COM91]} COMDISCO SYSTEMS,~INC. \hfil\break
  {\it Block Oriented Systems Simulator (BOSS) \hfil\break
      User's Guide} \hfil\break
      COMDISCO SYSTEMS, INC., BOS1080, Version 2.7, Jun.~1991 \hfil\break

\item{[Lev90]} S.~Levy, D.~E.~Knuth \hfil\break
  {\it The {\tt CWEB} System of Structured Documentation} \hfil\break
  \TeX\ 3.14 Distribution \hfil\break

\item{[Knu89a]} D.~E.~Knuth \hfil\break
  {\it The {\tt WEB} System of Structured Documentation} \hfil\break
  \TeX\ 3.14 Distribution \hfil\break

\item{[Knu84]} D.~E.~Knuth \hfil\break
  {\it Literate Programming} \hfil\break
  The Computer Journal~{\bf 27:2}, 1984, pp.~97--111 \hfil\break

}
-----------------------------------------------------------------------
```

For those which are not tired (bored?) already, this is how 'cubspl.w' starts:

```
-- cubspl.w -----------------------------------------------------------
@<Declare functions for |cubspl.c| @>=
  @<Declare |cub_spline()| @>@;
  @<Declare |zbrent()| @>@;
  @<Declare |spline()| @>@;
  @<Declare |splint()| @>@;

@ The following constants are exported to all other \Cee\ translation
units via \.{drs-pp.h}:

@<Define constants for |cubspl.c| @>=
#define  NATURAL  1.e30         /* Use this for |yp1|, |ypn| to
                                   specify ``natural'' boundary
                                   conditions in |spline()|. */
#define MAX_INTPTS   100        /* Maximum number of points for
                                   interpolation. */

@ The file \.{cubspl.c} contains:

@(cubspl.c@>=
@=/* This file (cubspl.c) has been produced automatically from 'drs-pp.w'@>@#
```

```
@=   and 'cubspl.w'. You almost certainly shouldn't edit it! */@>@#
  @<Local variables and constants for |cub_spline()| @>@;
  @<Perform cubic spline interpolation |cub_spline()| @>@;
        :
-----------------------------------------------------------------------
```

From: **Norman Ramsey**
Date: 18 Oct 1994

Felix Gaertner writes: I'm currently going over some code I wrote a year ago and came across an interesting point concerning literate programming style. I am interested in how people write the inclusion of header files `literally'? [distribute #include or write them all in one place?]

Both C and Modula-3 have the property that it is poor practice, or actually incorrect, to #include the same file (IMPORT the same interface) twice. Since it is difficult to protect against this error when the #includes are distributed, I tend to write them all in one place. I usually put this near the end of the file. #include is even uglier than you might expect since there are order dependencies (e.g. sys/types.h before other files).

From: **Jacob Nielsen**
Date: 19 Oct 1994

Norman Ramsey writes: Both C and Modula-3 have the property that it is poor practice, or actually incorrect, to #include the same file (IMPORT the same interface) twice. Since it is difficult to protect against this error when the #includes are distributed, I tend to write them all in one place. I usually put this near the end of the file. #include is even uglier than you might expect since there are order dependencies (e.g. sys/types.h before other files).

Normally I do like this:

```
"code.cc"=
  <Needed |#include|s for file |code.cc|>
  #include "code.h"
  <Methods in file |code.cc|>
```

All my code files (.cc --- yes C++ code) has header files (.h), so I write that literately in the file and all the other #includes gets mentioned in <Needed |#include|s for file |code.cc|>. If there are many #includes I then do a:

```
<Needed |#include|s for file |code.cc|>=
  <Include standard defs. in |code.cc|>
  <Include other defs. in |code.cc|>
```

This way I can find the #include's easily (IMHO) PS: to save myself from trouble, I always protect the .h files from being included more than once by using '#ifndef ... #define ... #endif'

From: **Aaron Digulla**
Date: 19 Oct 1994

Norman Ramsey writes: Both C and Modula-3 have the property that it is poor practice, or actually incorrect, to #include the same file (IMPORT the same interface) twice. Since it is difficult to protect against this error when the #includes are distributed, I tend to write them all in one place. I usually put this near the end of the file. #include is even uglier than you might expect since there are order dependencies (e.g. sys/types.h before other files).

Well, modern compilers have a flag to avoid multiple inclusion of the same file. For my own projects, I use a method that does always work: 1. All files are protected with a #ifndef \uppercase{filename}, ie. if the file is in include/ as "project/main.h", the #ifndef would be #ifndef PROJECT_MAIN_H and then a #define so the contents of the file are read only once. 2. All #includes are also protected with #ifndef's (not really neccessary, but makes it a bit faster). 3. Last but not least every include-file can be read at any time (even alone). If it needs any other files, it can #include them because of 1 and 2. So the sub-file "project/sub.h" contains

```
    #ifndef PROJECT_MAIN_H
    #   include <project/main.h>
    #endif
```

which solves all problems and makes life really easy (even more if you have an editor where you simply have to specify the name of

the file and it creates the #ifndef..#include..#endif for you :) "(to) optimize: Make a program faster by improving the algorithms rather than by buying a faster machine."

Several document types from the same noweb file?

From: **Fernando Mato Mira**
Date: 17 Oct 1994

Would anybody care to give a short example of how to write/use a noweb file to generate:

1. A reference manual (only function signatures / public slots of stuctures appear in the documentation + some description + short example)

2. A user's guide (no signatures, presentation follows a different order, long examples).

3. An implementors' manual (like a refence manual, but with discussion about caveats, etc.).

4. A classic literate program documentation (need not include the examples).

I guess the user's guide (#2) should probably be written separately.

By the way, is it possible to `include' pure documentation files into a noweb file?

a. Editing a file with correct code just do document seems a bad idea. An error might be introduced accidentally.

b. A documentation might be cleaned-up by someone not authorized to change the code.

c. Touching the file with the code is uncool with make et al.

d. Too much documentation mixed with the code makes it unpractical for programming (unless non-code sections can be hidden in Emacs).

PS: If what I want seems to point to some other system, please note that I program in Lisp.

From: **Norman Ramsey**
Date: 18 Oct 1994

Fernando Mato Mira writes: Would anybody care to give a short example of how to write/use a noweb file to generate:

1. A reference manual (only function signatures / public slots of stuctures appear in the documentation + some description + short example)

2. A user's guide (no signatures, presentation follows a different order, long examples).

3. An implementors' manual (like a refence manual, but with discussion about caveats, etc.).

4. A classic literate program documentation (need not include the examples).

Don't even *think* about it. All the experiments I have tried along these lines have ended in disaster. The problem is, how are you going to edit and understand the source code for 4 different documents simultaneously? Especially such different documents? Slipping pieces of a reference manual into a document that is mostly a classic literate program seems OK. That is, it's not too hard to get the manual right, and the intercalated chunks aren't too disruptive. But even that is a bit creaky.

By the way, is it possible to `include' pure documentation files into a noweb file?

This question suggests that maybe you don't "get" the literate-programming paradigm. I suggest you read some of Don Knuth's papers, and also the Don Lindsay literate programming column from CACM, with special attention to reviewer Harold Thimbleby's comments.

a. Editing a file with correct code just do document seems a bad idea. An error might be introduced accidentally.

Leaving your documentation off in another file seems a bad idea. It might get *way* out of date.

b. A documentation might be cleaned-up by someone not authorized to change the code.

The literate document *is* the program. Someone not authorized to touch it ought not to touch it.

c. Touching the file with the code is uncool with make et al.

Use cpif.

d. Too much documentation mixed with the code makes it unpractical for programming (unless non-code sections can be hidden in Emacs).

If the documentation isn't helping you program it's not doing its job.

From: Barry Schwartz
Date: 19 Oct 1994

Fernando Mato Mira writes: Would anybody care to give a short example of how to write/use a noweb file to generate:

1. A reference manual (only function signatures / public slots of stuctures appear in the documentation + some description + short example)

2. A user's guide (no signatures, presentation follows a different order, long examples).

3. An implementors' manual (like a refence manual, but with discussion about caveats, etc.).

4. A classic literate program documentation (need not include the examples).

I have argued (in this newsgroup, in fact) that there are literary reasons not to do this. My argument is that the manuals are works of literature on the same level as the literate program, and so should be written separately. There can be information about how to use the program written into the literate program itself, but good exposition style dictates that this information be terse. Notice that I make these arguments on grounds that come from expository writing and not traditional programming practice; I believe that literate programming is as much a task of expository writing as it is a task of producing a working program. Actually I view all serious programming this way (read "The Elements of Programming Style," by Kernighan and Plauger), but literate programming gives you better tools to do the job.

An "implementor's manual" is an interesting thing to contemplate. The code itself should be documented in the literate program, and so there is no need for a separate manual. If you need one, then you haven't written an adequate literate program; go back and edit it. Designs present another obstacle, however. If the design is written in a precise language (say a variant of Ada, which I believe is used by IBM), then it should be possible to use noweb. Then you have at least two literate programs, one for the design and one for the code. Together, and along with the Requirements, they make up an implementor's manual, as well as the working program.

By the way, is it possible to `include' pure documentation files into a noweb file?

That's like asking if it is possible to love someone without commitment. The question shouldn't be "is it possible," but "is it valuable." "Is it valuable to love someone without commitment?" The answer is No. Similarly, there is little value in including the separate documentation within the noweb file. You might as well leave it separate. If you want to attach the documentation to the code in some secure way, you might as well put them together in a zip archive or something like that. You will gain none of the benefits of literate programming, of course.

a. Editing a file with correct code just do document seems a bad idea. An error might be introduced accidentally.

Code is not really correct unless you can edit it without great fear of breaking it. Most code has to be maintained, after all. Read "The Elements of Programming Style," by Kernighan and Plauger.

b. A documentation might be cleaned-up by someone not authorized to change the code.

This is a program configuration problem, not a problem with noweb. If this can happen then there is something wrong with the quality management process.

c. Touching the file with the code is uncool with make et al.

The cpif program addresses this problem.

d. Too much documentation mixed with the code makes it unpractical for programming (unless non-code sections can be hidden in Emacs).

The last thing you want to do is ignore the explanation of what the code does, when you go in to modify the code. In a literate program, ideally the exposition and the code are an organic entity. This is not like what you might imagine, the code as "sufficient explanation of what's going on" and the documentation as "here's a few notes about what I've written." The way I view things, you should have the plain language notes there to keep you on course, and if you find something is wrong you should update the plain language along with the code. After all, the program is not correct unless both documentation and code are correct. (Read "The Elements of Programming Style," by Kernighan and Plauger.)

Furthermore, if you really find that mixing docs and code is unpractical, then I think something is wrong, and I think I know what. The only way I can imagine this happening is if the program has bad modularity, in either the programming sense or the literate sense. There is some question as to how well Emacs modes handle literate programming, but that's an Emacs problem, not a literate-programming problem.

PS: If what I want seems to point to some other system, please note that I program in Lisp.

I agree with the view that a literate programming system should be language-independent. Noweb is such a system. Lisp is no problem for it.

From: **Kasper Osterbye**
Date: 22 Oct 1994

Fernando Mato Mira writes: Would anybody care to give a short example of how to write/use a noweb file to generate:

1. A reference manual (only function signatures / public slots of stuctures appear in the documentation + some description + short example)

2. A user's guide (no signatures, presentation follows a different order, long examples).

3. An implementors' manual (like a refence manual, but with discussion about caveats, etc.).

4. A classic literate program documentation (need not include the examples).

*Norman Ramsey writes: Don't even *think* about it. All the experiments I have tried along these lines have ended in disaster. The problem is, how are you going to edit and understand the source code for 4 different documents simultaneously? Especially such different documents? Slipping pieces of a reference manual into a document that is mostly a classic literate program seems OK. That is, it's not too hard to get the manual right, and the intercalated chunks aren't too disruptive. But even that is a bit creaky.*

Well, but the problem remains then. Literate programming is well suited to make sure that the program and its low level exposition are kept up to date, but how can we ensure that the other documents Fernando mentions has the same quality? You say that you have attempted it in many different ways - are there no hope to extend the literate paradigm to include the apparently natual thing Fernando wants to do?

From: **Eric van Ammers**
Date: 24 Oct 1994

Kasper Osterbye writes: Well, but the problem remains then. Literate programming is well suited to make sure that the program and its low level exposition are kept up to date, but how can we ensure that the other documents Fernando mentions has the same quality? You say that you have attempted it in many different ways - are there no hope to extend the literate paradigm to include the apparently natual thing Fernando wants to do?

A relavant paper about this problem appeared recently: Jim Welsh, Jun Han, Software Documents: concepts and tools, Software-Concepts and Tools (1994) 15: 12-25

From: **Norman Ramsey**
Date: 28 Oct 1994

Kasper Osterbye writes: Well, but the problem remains then. Literate programming is well suited to make sure that the program and its low level exposition are kept up to date, but how can we ensure that the other documents Fernando mentions has the same quality?

You say that you have attempted it in many different ways - are there no hope to extend the literate paradigm to include the apparently natual thing Fernando wants to do?

I think extending the literate paradigm is a misguided idea. Literate programming is about writing documents that are also programs. The mechanisms commonly used to create literate programs also appear to be capable of combining multiple documents into a single source, but that's an accident, and we shouldn't let it seduce us into insisting on doing things that way. In fact, literate programming as practiced today is really good only for producing single documents.

I know of two experiments that are not failures. One is one of mine, which includes portions of a reference manual (as *code*) in a document that is the implementation of the tool described in the reference manual. This scheme is workable only because the size of the reference-manual fragments is a tiny fraction of the whole document, well under 10%. (The bulk of the manual lives in its own source files anyway.) I would not call it a success, but it's not a failiure, either. I'm not sure the benefit we get outweighs the added difficulty of editing the source. I may be able to tell better as we see how the implementation and reference manual evolve over time. I certainly wouldn't advocate that other people do this except as an experiment.

The one experiment I know of that was successful was that of Jon Bentley and others at Bell Labs in describing the PRL5 language. In that case: a) Six or so documents were generated from the same source. b) There was no code involved, only documentation. c) The six documents had identical outlines (structures). d) (I believe) documents didn't share text. For this limited problem, they got good results using a custom little language to generate the documents. I see no way of generalizing to documents that have different structures.

I've seen claims for hypertext systems, but I'm not sure hypertext is a 'document' in the sense I'm used to thinking of. I have found hypertext useful for browsing to find bits of information, but much less so for reading in bulk to get a coherent, in-depth understanding of anything. I'm also not sure what it means to have *multiple* documents if everything's connected together in a hypertext web.

As for where we go from here, I think people ought to be thinking about new tools and techniques to support the creation of multiple documents with different contents and structure but which have to be kept consistent. It's a hard problem, and I'm not sure the skills of the literate programmer are a great match for it.

WEB for Turbo Pascal

From: **Denis Roegel**
Date: 19 Oct 1994

Is there a WEB working with Turbo Pascal?

From: **Roland Kaufmann**
Date: 20 Oct 1994

In my view, Turbo Pascal is a particular implementation of (a compiler for) the programming language Pascal, for which the original (Donald E. Knuth's) WEB was developed. Without wanting to offend anyone, I consider using a non-portable, non-standard language like ``Turbo'' Pascal somewhat at odds with the concept of literate programming. I haven't used Turbo Pascal for a long time, but the old versions I used many years ago always annoyed me with their concept of fast compilation, quick fix of the first (syntax) error and continuing --- I do not think that reliable programs should be written like this, regardless whether they are literate or not. By the way, does anyone out there where I could get a copy of Brian Kernighan's article ``Why Pascal is not my favorite language''?

From: **Lee Wittenberg**
Date: 20 Oct 1994

The original WEB should work with Turbo Pascal (excluding extensions, but @f can help with that). Otherwise, the language-independent systems (noweb, nuweb, CLiP, etc.) will all work with Turbo. I've been doing some work in Turbo with noweb recently, which will eventually get posted to my samples.LP area. This stuff includes a filter that rearranges type definitions so that you can define them naturally:

```
@ Explanations for [[SomeArrayType]]
<<Type declarations>>=
 SomeArrayType = ARRAY[1..whatever] OF Basetype;
@ Explanations for [[Basetype]]
<<Type declarations>>=
 BaseType = ...
```

without having to kludge things up so that [[Basetype]] will be defined before [[SomeArrayType]] as Pascal requires. The filter is a

post-processor, so it should work with any literate programming system; it's a hack, so there are no guarantees (but it's a literate program hack, so I won't be surprised if it does work for others).

C/C++ -> TeXinfo/HTML/LaTeX/nroff -man

From: **Oliver Imbusch**
Date: 21 Oct 1994

I'd like to have a tool for generating documentation from C/C++ source code. Features should be as follows:

1) source code and documentation go to one file

2) input is pure C/C++, ie., documentation stuff is held by comments, and no tangle-like source code extraction is needed

3) output is preferably TeXinfo or HTML, ie., a hypertext system; LaTeX and nroff -man could be supported as well

4) both user's and implementator's aspects of documentation are covered

5) very small interface: text between /// and newline (or /** and */, resp.) appears in the user's documentation, text between //// and newline (or /*** and */, resp.) additionally appears in the implementator's documentation

6) cross-references are generated automatically: There's a node for each class, and a sub-nodes for each method; cross-references go to super- and sub-classes (which are known if a whole class library is scanned), and friends. Usage graph (calling/called by). Index.

While there are lots of WEB-like tools, they have some major disadvantages:

1) Input is not pure C/C++ but a meta-language, and some sort of weave/tangle tools are needed.

2) Most tools are overfeatured, ie., due to their flexibility and language independence there is a large set of (IMHO) superflous commands. For example, if a tool knows about the language you don't need commands for cross referencing. It can be done automatically.

While there's a package (Graham Stoney's c2man) that nearly perfectly meets the minimal interface requirements (and the user's/implementator's differentiation should easily be added), it still has no C++ support, and lacks a true hypertext backend.

Does anybody know whether there's a tool that provides some of the features mentioned above and that could be used as a base for enhancement? It should be quite difficult not to use a full-blown C++ parser. Would you suggest to use cppp? Or could it be useful (possible?) to add all that stuff to gcc? Any hints/suggestions/contributions are welcome. PS: Here's a sample C++ file. Note that neither the code nor the documentation makes any sense, and many comments would not appear in a real program (those meta-documenting the documentation). PPS: The project's working title is DoX (DOcumentation eXtracter).

```
// This is a two-slash comment. It will be ignored
/* This is a one-asterisk comment. It will be ignored */

/*
Three/four-slash comments are what DoX deals with, as well as
two/three-asterisk comments. DoX will ignore comments with more
slashes/asterisks (decoration):
*/

/// This will go to the user's guide (three slashes).
/** This too (two asterisks). */

//// This goes to the implementators guide (four slashes).
/*** This too (three asterisks) */

// This and the following line will be ignored:
////////////////////////////////////////////////////////////////

/* This and the next line aren't used, either. */
/**************************************************************/

// Comments at the beginning of a file give a description of a group
// of related classes (cluster). All classes defined in files in one
// sub-directory could be also considered as cluster. Clusters could
// be the base for structuring documents in a
// chapter/section/... manner, whereas the class graph cannot since it
// does not form a tree in the general case.
```

```
/// The following classes provide a nice interface to the foobar
/// package.

// Comments in front of a class (no newline) give a genral overview.
// Within a class declaration, there should be no further
// documentation for methods, but the meaning of variables and
// nested classes is mentioned.

/// Class A deals with foo stuff.
class A
{
  public:
    A (int a);
    ~A ();

  protected:
    protA (int a);

  private:
    A (int a, int b);
    int privA ();

    // Note: since we are in the private class section, documentation
    // only goes to the implementator's guide, regardless wether we are using
    // 3 or 4 slashes.

    int _a;  //// holds the number of a's
    int _b;  ///  holds the number of b's
};

//////////////////////////////////////////////////////////////////////

// Within an implementation section, methods can be grouped using
// comments surrounded by empty lines:

/// Constructors/destructor

/// Constructs A from one int.
A::A (int a /** initial value */ ) { _a = a; }

/// Construct A from two ints.
//// The assignment to _a must come before the assignment to _b.
A::A (int a, /// first value
      int b  /** second value */)
{ _a = a; _b = b; }

/// Destroy.
A::~A () {}

//////////////////////////////////////////////////////////////////////

/// Selectors

/// Returns a.
int A::privA () { return _a; }

//////////////////////////////////////////////////////////////////////

/// Modifiers

/// Assignment (int)
A::protA (int a) { _a = a; }

/// B adds bar features to A and thus makes it a real foobar.
class B : public A
{
  public:
```

```
    B (int b);
    ~B ();

  private:
    int _b;
};

/// Constructors/destructor

/// Contruct from int.
//// FIXME: magic number should be replaced.
B::B (int b /** initial value without offset */) : A (b + 6060842), _b (b) {}

/// Destroy.
B::~B () {}

////////////////////////////////////////////////////////////////////

/// Supporting functions.

/// The qsort subroutine sorts a table of data in place. It uses the
/// quicker-sort algorithm.
//// The quicker-sort algorithm is as follows: ...
//// See also Hoare: "..."; Knuth: "The art of computer programming"
void qsort
(
    void*  base,     /// pointer to first element
    int    nElems,   /// number of elements
    int    elemSize, /// size of each element
    /** pointer to comparison function */
    int    /// <0:  lhs <  rhs
           /// ==0: lhs == rhs
           /// >0:  lhs >  rhs
           (*compare) (const void* lhs, /** left hand side of comparison */
                       const void* rhs  /** right hand side of comparison */)
)
{}
```

From: **Norman Ramsey**
Date: 24 Oct 1994

While there are lots of WEB-like tools, they have some major disadvantages: 1) Input is not pure C/C++ but a meta-language, and some sort of weave/tangle tools are needed.

Not everyone thinks using a meta-language is a disadvantage. It's not clear why you'd be willing to use a tool to produce documentation but not willing to use a tool to produce source code. Make is good at automating such tasks.

2) Most tools are overfeatured.

Too true. Check out noweb and nuweb. noweb has the smallest command set of any of the LP tools. Both have some HTML support; I'm not sure which is better in that regard. noweb has some automatic cross-reference support for C, but it doesn't really work for C++.

Does anybody know whether there's a tool that provides some of the features mentioned above and that could be used as a base for enhancement?

You might try to extend noweb if you can overcome your dislike of tangle-like tools. Noweb's author claims it is easy to extend. Some users seem to agree. Where are all the standalone cross-reference tools we had in the 70s? *sigh*

From: **Lee Wittenberg**
Date: 24 Oct 1994

*Oliver Imbusch writes: 5) very small interface: text between /// and newline (or /** and */, resp.) appears in the user's documentation,*

*text between //// and newline (or /*** and */, resp.) additionally appears in the implementator's documentation*

I feel it's necessary to point out that your ///, /**, etc. conventions are also a "meta-language," but then again, "a foolish consistency is the hobgoblin of little minds." ;-)

What is cpif?

From: **Sven Utcke**
Date: 21 Oct 1994

Fernando Mato Mira writes: c. Touching the file with the code is uncool with make et al.

Barry Schwartz writes: The cpif program addresses this problem.

This is now the second time that I hear about cpif. What is it? Can it be used with, say, FWEB (the literate program of my choice)? If so, where can I get it (ask archie, I suppose?)?

From: **Joachim Schrod**
Date: 21 Oct 1994

Sven Utcke writes: This is now the second time that I hear about cpif. What is it?

cpif is part of the noweb distribution, it copies standard input to a file only if the contents did not change. (Or did change, as you want.)

```
---------------- snip snap ----------------------------------------
#!/bin/sh

# cpif [ -eq -ne ] file...
# copy standard input to each of the named files
# if new * old is true or old doesn't exist;
# * defaults to -ne

PATH=/bin:/usr/bin

# set -x
op=-ne
case "$1" in
-eq|-ne)        op=$1; shift ;;
-*)             echo 'Usage: '`basename $0`' [ -eq -ne ] file...' 1>&2; exit 2
esac
case $# in
0)              echo 'Usage: '`basename $0`' [ -eq -ne ] file...' 1>&2; exit 2
esac

new=/tmp/$$
trap 'rm -f $new; exit 1' 1 2 15        # clean up files

cat >$new
for i
do
        cmp -s $new $i
        case $op$? in
        -eq0|-ne1|*2)   cp $new $i
        esac
done
rm -f $new
---------------- snip snap ----------------------------------------
```

Actually, not long ago I needed to extend this to `real' file copy operations, so here is my copyif. (E.g., I use it for parser generators like

PCCTS who create the scanner as an intermediate file and would trigger spurious compilation of the scanner as it has changed.) As it's roughly upward compatible (as long as only one file is `cpif'ed), I use it for noweb, too... :-)

```
---------------- snip snap ----------------------------------------
#!/bin/sh
# $ITI: copyif,v 1.1 1994/07/14 12:55:24 schrod Exp $
#-----------------------------------------------------------------------

#
# copyif -- copy files if comparison succeeds
#
# (history at end)

# SYNOPSIS
#
#       copyif [-eq | -ne] file
#       copyif [-eq | -ne] file1 file2
#       copyif [-eq | -ne] file ... dir
#
#
# copyif copies a file if a comparison succeeds or if the target file
# does not exist. The comparison is either a test if the contents is
# equal (-eq) or if it has changed (-ne). The default is -ne.
#
# The first call form copies standard input to <file>file</>. The second
# one copies <file>file1</> to <file>file2</>. The third copies all
# <file>files</>s into directory <file>dir</>.
#
# The source file(s) and also the target file/dir must be readable,
# and target must be writable.
#
# A typical usage is in Makefiles where a file shall only be copied if
# it has changed.
#
#

cmd=`basename $0`
usage()
{
        cat <<_EOF_ >&2
usage:
        $cmd: [-eq | -ne] file
        $cmd: [-eq | -ne] file1 file2
        $cmd: [-eq | -ne] file ... dir
_EOF_
        exit $1
}

# parse options...

test $# = 0  -o  "$1" = '-?'  &&  usage 1

# default is -ne

case "$1" in
    -ne|-eq) test=$1
        shift
        ;;
    *)  test='-ne'
        ;;
esac

# one parameter => save source from stdin in $tmp_file
if [ $# = 1 ]
   then tmp_file=/tmp/copyif$$
        trap "rm -f $tmp_file" 0 1 2 3 15
        cat >$tmp_file
```

```
        src=$tmp_file
        dest_file=$1
        dest_dir=''
fi

# two parameter
if [ $# = 2 ]
   then src=$1
        # Target argument might be a directory, ie, this might be a
        # call of form 3.
        if [ -d "$2" ]
           then dest_file=''
                dest_dir=$2
           else dest_file=$2
                dest_dir=''
        fi
fi

# n parameter
#       $1 .. $n-1 are $src
#       $n is $dest_dir
if [ $# -gt 2 ]
   then # Split list of arguments on last blank and assign thereby the
        # source files to $src.
        src=`expr "$*" : '\(.*\) '`
        dest_file=''
        # Discard all source arguments
        shift `expr $# - 1`
        # Check if target argument is a directory, complain if not
        if [ ! -d "$1" ]
           then echo "$cmd: $1: not a directory." >&2
                usage 2
           else dest_dir=$1
        fi
fi

# pre:
# $src is list of source files
# either $dest_dir is destination directory
#     or $dest_dir is empty and $dest_file is destination file and
#        $src is only one file

result_code=0

for file in $src
   do   test "$dest_dir"  &&  dest_file=$dest_dir/$file
        # now $dest_file holds the target file name
        # check if it exists
        if [ -f $dest_file ]
           then cmp $file $dest_file >/dev/null
                result="$test$?"
                case "$result" in
                    *2) # error in cmp command, message was issued
                        exit 2
                        ;;
                    -eq1|-ne0)
                        continue
                        ;;
                esac
        fi
        cp $file $dest_file
        result_code=`expr $result_code + $?`
   done

# If there was at least one error in cp, add 10 to the result code.
# This way we get a unique exit code that we can distinguish from
# internal errors.
```

```
test $result_code != 0  &&  result_code=`expr $result_code + 10`
exit $result_code

#============================================================================
#
# $ITIlog: copyif,v $
# Revision 1.1  1994/07/14  12:55:24  schrod
#     Initial revision.
#
---------------- snip snap --------------------------------------
```

From: **Lee Wittenberg**
Date: 24 Oct 1994

Sven Utcke writes: This is now the second time that I hear about cpif. What is it? Can it be used with, say, FWEB (the literate program of my choice)? If so, where can I get it (ask archie, I suppose?)?

cpif is a tool that comes with the noweb distribution. It's designed to work with makefiles, and should work with any system. You can find it in the noweb distribution in the CTAN archives (under the shell directory; the man page cpif.1 is in the xdoc directory). Cpif is a shell script, but Barry Schwartz has ported it to DOS, but this version hasn't been officially released yet, because Lee Wittenberg has been to d--- lazy to build a new DOS noweb distribution with it enclosed.

How do you write (small) classes with CWEB?

From: **Giovanni Pensa**
Date: 26 Oct 1994

Well, I have a question... I program in C++ and I use CWEB. No real problems, but... I'm not sure if literate programming is very useful with object oriented programming. I think (or just feel) that some features were useful years ago, with a standard Pascal or a bare C. (Ops... will the great DEK kill me for a sentence like this?)

Literate programming can help with global data? A C++ program shouldn't have global data. With literate programming you can declare your variables where you want? Also with C++. (the problem is that with CWEB's philosophy, the variables are moved at the beginning, sigh) (etc.) You know, I'm not against WEB. I'm just saying that some things are useless (and sometimes are a real problem) and that I might need other features. (I don't know what yet.)

```
@* Point.
This is a useless Point class.

@<The |Point| class@>=
class Point
{
        int x, y;
public:@/
        Point();
        Point( Point& );
        Point( int, int );
};
@<|Point|'s functions@>@;

@ This is the first constructor.
@<|Point|'s functions@>=
inline Point::Point()
{
        x = y = 0;
}

@ This is the second constructor.
@<|Point|'s functions@>=
inline Point::Point( Point& p )
{
        x = p.x;
```

```
        y = p.y;
}
```

That's more or less how I work. The things that I don't like: 1) I'd like "Point& p" more than "Point &p". 2) I'd like for CWEB to put a new line after "public:". 3) ... (Yes, sometimes I create bigger functions...)

And, in general, in this way literate programming is not really useful. The functions are small "di per se", they are "inline", they are logically under the class, they tell a lot just by their name, ... Ok, I know I'm confused and confusing. I just want to know your opinion. "How do you write (small) classes with C++ and CWEB?" "Is your style better than mine?" "Do you think there are some problems, too?"

P.S. I still don't believe that I'm writing >against< DEK! Donald, will you ever forgive me? (I know you don't read email...)

From: **Marc van Leeuwen**
Date: 26 Oct 1994

Giovanni Pensa writes: I'm not sure if literate programming is very useful with object oriented programming. I think (or just feel) that some features were useful years ago, with a standard Pascal or a bare C.

So why worry? You don't _have_ to use any features just for their sake.

Literate programming can help with global data? A C++ program shouldn't have global data. With literate programming you can declare your variables where you want? Also with C++. (the problem is that with CWEB's philosophy, the variables are moved at the beginning, sigh)

Although you _can_ declare variables anywhere you want, it is preferable to declare them as local as possible wherever you can. In C this is usually possible, in Pascal only if you are writing very small functions. So WEB can help you out in difficult cases, but please don't misuse this feature to write unnecessarily bad programs.

The things that I don't like: 1) I'd like "Point& p" more than "Point &p". 2) I'd like for CWEB to put a new line after "public:".

So (unless you want to keep typing explicit format controls all the time) you want to modify the grammar of CWEAVE. A problem, and in my opinion the most serious problem, with (Levy/Knuth) CWEB is that its grammar is so hard to understand, and therefore to modify. Another problem is specific to C/C++, namely that it's syntax for declarations is so horrible. I like to write declarations like |char* s="string";| myself, but the sad fact is that in C a declaration is a type specifier followed by an _expression_, and the |*| belongs to the expression syntactically, not to the type specifier; this makes it almost impossible to specify rules that separate the |*| from the |s| in my example and are still capable of handling the general case properly. Now I believe the use of |&| in C++ declarations is more limited than that of |*|, so if you make |&| into a special syntactic category, you might have a better chance of getting your first wish satisfied. Your second wish should be easy: rule 90 reads |public_like colon -> tag|, and if you include a forced break at the end of the translation for this rule, you should get what you want.

And, in general, in this way literate programming is not really useful. The functions are small "di per se", they are "inline", they are logically under the class, they tell a lot just by their name, ...

If your program consists mainly of such functions, then chances are it is not doing much algorithms, just a lot of administration (in my opinion this is the area where object oriented programming is most applicable). I have written such a program myself (a small window application) and found that the benefits of literate programming are not very great here, mainly because there is so little to say about the code that is not immediately obvious. Nevertheless there is no reason not to use CWEB in such cases; it still makes your code and the few comments you do have look a lot nicer. In any case there is no obligation to break up a small function into multiple modules just because you are trying to practice Literate Programming.

Donald, will you ever forgive me? (I know you don't read email...)

You can be pretty certain that Don Knuth doesn't read this news group. Nor do I think he objects to people using literate programming in a different way than he does himself.

From: **Jacob Nielsen**
Date: 26 Oct 1994

Giovanni Pensa writes: Well, I have a question... I program in C++ and I use CWEB. No real problems, but... I'm not sure if literate programming is very useful with object oriented programming. I think (or just feel) that some features were useful years ago, with a standard Pascal or a bare C. (Ops... will the great DEK kill me for a sentence like this?)

I agree that with the small functions/methods normally associated with object oriented programming cannot take benefit from inlining of scraps, but the ability to write the code and the explanation close to each other is still very usefull (C++ is not that expressive :-) Also the explanation of the grand scheme of things is better done using a typesetting language or system than writing comments in a code file (I know that my comments in code files are in general very small and inadequate.) There is no doubt that PASCAL, C etc. benefits more from literate programming than the newer programming languages, but the need for literate programming has not disappeared.

Literate programming can help with global data? A C++ prog shouldn't have global data.

Ah, but they do sometimes :-) IMHO, global data are evil in any language.

With literate programming you can declare your variables where you want? Also with C++. (the problem is that with CWEB's philosophy, the variables are moved at the beginning, sigh)

The joy of CWEB. Anyway, why is that a problem --- you use *short* and clear scraps, no?

The things that I don't like: 1) I'd like "Point& p" more than "Point &p". 2) I'd like for CWEB to put a new line after "public:". 3) ... (Yes, sometimes I create bigger functions...)

The joy of prettyprinting :-)

And, in general, in this way literate programming is not really useful. The functions are small "di per se", they are "inline", they are logically under the class, they tell a lot just by their name, ...

So, why split them up in the first place?

Ok, I know I'm confused and confusing. I just want to know your opinion. "How do you write (small) classes with C++ and CWEB?"

I place logically related methods in the same scrap, e.g. for simple classes I place the constructors in the same scrap.

"Is your style better than mine?" "Do you think there are some problems, too?"

My primary concern is if my programs are well enough explained. Disclaimer: I have never used CWEB --- I fell in love with nuweb (noweb is nice too)

From: **Timothy Larkin**
Date: 26 Oct 1994

Giovanni Pensa writes: I'm not sure if literate programming is very useful with object oriented programming.

I am currently working on an application in C++ using LP. I find it very useful. I have a standard approach that works like this:

```
@* The name of a class.
@(foo.h@>=
#include "baseclass.h"

class Foo : public baseclass {
private:
   @<private declarations@>@;
public:
   @<public declarations@>@;
```

```
protected:
   @<protected declarations@>@;
};

@ @c
#include "foo.h"
@<includes@>@;
@<methods@>

@ Destructors and constructors.
@<public...@>=
Foo();
~Foo();

@ @<methods@>=
Foo::Foo()
{
   @<construction@>@;
}

Foo::~Foo()
{
   @<destruction@>@;
}

@ Now I proceed on an exposition that often centers on member variables.
For instance, I use an array |frob| like this. LP allows me to collect the
diverse declartions into a single place.
@<private...@>=
char *frob;

@ @<construction@>=
frob = new char[80];

@ @<destruction@>=
delete [] frob;

@ Define some access functions.
@<public...@>=
const char *GetFrob();
void SetFrob(char*);

@ @<methods@>=
const char *Foo::GetFrob()
{
   return frob;
}

void Foo::SetFrob(char *)
{
...etc.
```

Enough of that. You get the idea. I find that literate programming pays off handsomely in object oriented programming. If I need to make a change to |frob|'s contract, I know that I will find it spelled out in one section of text. I don't have to jump around between include files and main body, constructors, destructors, access functions, etc. Furthermore, when I develop classes that are closely linked with each other, I can easily write the code in a single WEB, interdigitating code from different classes.

```
@ @<Foo methods@>=
void Foo::DoSomething() {...}

@ This implies that |Bar| must do something else.
@<Bar methods@>=
void Bar::DoSomethingElse() {...}

@ Now |Foo| can go ahead with the final act.
void Foo::FinalAct() {...}
```

I think that literate programming and object oriented programming are made for each other, even more perhaps than literate

programming and C. C++ is such a clumsy tool: it needs all the help it can get!

From: **Barry Schwartz**
Date: 28 Oct 1994

Marc van Leeuwen writes: You can be pretty certain that Don Knuth doesn't read this news group. Nor do I think he objects to people using Literate Programming in a different way than he does himself.

Neither (says he) does he object to people using literate programming tools other than WEB and CWEB. If hypermedia are "perfected", then that may take care of a lot of objections that some object oriented programmers have towards literate programming. I personally view linear text as the most compelling form of exposition, but perhaps sometimes exposition has to make some compromises.

From: **Kayvan Sylvan**
Date: 29 Oct 1994

I don't use CWEB, so I won't comment on the CWEAVE-specific problems you were griping about. The way I do C++ code in nuweb is best illustrated by an example. A few general points: I always put one logical module (interface and implementation of a class or set of related classes) in one web file. I have a method that allows me to integrate all the WEB files in my project into one document (with integrated indices) or just print that one module (while I'm developing it). So, I get the expository benefits of a huge monolithic web and the development benefits of having small web files that correspond to my program modules. Here's a sample:

```
\section{The World Class}

The {\tt World} class contains most of the objects in the simulation.
Let's look first at the interface to this class.

\subsection{World Interface}

Here is an overview of the interface to {\tt World}.

@o World.h
@{// This may look like C, but it's really -*-C++-*-
// Copyright (C) 1994, Kayvan Sylvan, Sylvan Associates. All rights reserved.
//
#ifndef __WORLD_H__
#define __WORLD_H__
#include "Thing.h"

class World {
public:
  World();                      // constructor
  ~World();                     // destructor
  void add(Thing*);             // Method for adding Thing objects.
  void tick();                  // One unit of time passes.
  void display();               // Show the user what the world looks like.
private:
  @<World Private Parts@>
};
#endif
@| World::add World::tick World::display @}

\subsection{World Implementation}

The implementation of {\tt World} includes the definition of the class
private parts as well as the function definitions that make up the
class.

@d World Private Parts
@{
Thing** list;                   // List of Thing object pointers.
Map map;                        // Map of the world
@| list map @}
```

```
Now, the implementation of our {\tt World}.

@o World.cc
@{// Copyright (C) 1994, Kayvan Sylvan, Sylvan Associates. All rights reserved.

#include "World.h"

@<World Constructor@>
@<World Destructor@>
@<World add(Thing*) code@>
@<World tick()@>
@<World display@>
@}
```

In this code, as in all my class implementations, the first thing that I write about is the interface. The class's private parts are hidden away and expanded later. Sometimes, I even spend many pages talking about the public members, including various sample usages, before talking about the private members. My rationale is that the published interface (from the standpoint of the user/developer) is more important than the details of the implementation, and it also provides a nice reference section (for myself or others). The nice thing is that the header files that are generated also re-enforce this (by listing public members first).

The code that implements the class is put in its own file, with various pneumonic scrap names. All in all, I find that literate programming makes it very easy for me to write good object oriented programs that are well documented as a side effect. One caveat: literate programming does not take the place of a good object oriented analysis and design for your system, it merely supplements it. Hope you find this useful. "The trust and respect of a child is an honor to be earned, not demanded."

From: **Lee Wittenberg**
Date: 31 Oct 1994

Giovanni Pensa writes: Well, I have a question... I program in C++ and I use CWEB. No real problems, but... With literate programming you can declare your variables where you want? Also with C++. (the problem is that with CWEB's philosophy, the variables are moved at the beginning, sigh)

In this case, I think you're mistaken. Although the variables are moved to the beginning, this is only in the tangled output, which is meant to be read by a compiler, rather than a human being. Humans should be concerned only with the variables' locations in the woven web.

The things that I don't like: 1) I'd like "Point& p" more than "Point &p".

You need to modify CWEAVE's production rules to do this (it isn't as hard as it looks, but it ain't easy, either). Personally, I agree with your preference, but I see it as similar to the big-endian vs. little-endian debate: Both sides have supporting arguments, but if we could just flip a coin and pick one as a "standard," we'd save a lot of time (and money).

2) I'd like for CWEB to put a new line after "public:".

Again, you can modify the productions. This one should be quite easy.

Ok, I know I'm confused and confusing. I just want to know your opinion. "How do you write (small) classes with C++ and CWEB?"

Pretty much the same way I write large ones. A skeleton:

```
class blah {
        @<|blah|'s private members@>@;
public: @/      @q Yes, I prefer this, too @>
        @<|blah|'s public members@>@;
};

And so on...
```

"Is your style better than mine?"

Just different.

"Do you think there are some problems, too?"

Not major ones.

From: **Brian Danilko**
Date: 06 Nov 1994

Now for some useful literate programming discussion. I have been watching the talk on using literate programming with object oriented languages. My two cents worth: I have definitely noticed benefits using a literate programming meta-language (noweb in my case) while doing C++ programming. The current thing that I find useful (it changes depending where I am in the development cycle) is the ability to group information about a class in an intuitive, maintainable way. The user interface and the implementation close by but distinct without regard for efficiency specifiers (inline) or information hiding for users of the class.

IMHO C++ has two problems with its encapsulation model the class: 1) efficient inline implementation must be visible to a class user's compiler and 2) private, implementation data definitions must also be visible to a class user's compiler. As I tend to think of classes as abstract data types where the user must not make ANY assumptions about the implementation methods or data I tend to separate the information into different header files. The user can still look at implementation details but at least the separation clearly marks what information they should make assumptions on.

This separation though into separate files introduces a maintenance nightmare. Especially if you change methods from inline to outline and back. Enter literate programming with 'chunks' in a single file that can be output to multiple files. I now keep all class information in a single web, but direct output so that implementation information is separated from public information. Lets face it, object oriented programming can be done in C but the tools for encapsulation (and other things) are clumsy. C++ introduced the class keyword to make encapsulation easier. Well with a meta-language with 'chunks' we now have a tool that makes encapsulation even easier. That's just one benefit, but I believe an important one. I have found other benefits (and a certain elegance) to literate programming but as this post is already long, I will leave them to another day.

Header file creation in CWEB

From: **Ender Olcayto**
Date: 04 Nov 1994

I'm trying to write a large package in C and I'm using CWEB to do this. What I want is a global header file that will be common to several C source files, but I also want this documented. Unfortunately, when I use CWEB and the @) [or is it @(- I can't remember] control code to write to the .h file, ctangle also creates a .c file. Why can't I just tell ctangle what my desired output file is rather than writing to a .h file within my .w file? Well, I also want to write a Makefile with general rules .w:.c and .w:.h . Do I always have to treat a global header file as a specific case? PS. The global header file only needs to include #define statements.

From: **Marc van Leeuwen**
Date: 04 Nov 1994

Ender Olcayto writes: I'm trying to write a large package in C and I'm using CWEB to do this. What I want is a global header file that will be common to several C source files, but I also want this documented. Unfortunately, when I use CWEB and the @) [or is it @(- I can't remember] control code to write to the .h file, ctangle also creates a .c file. Why can't I just tell ctangle what my desired output file is rather than writing to a .h file within my .w file? Well, I also want to write a Makefile with general rules .w:.c and .w:.h . Do I always have to treat a global header file as a specific case? PS. The global header file only needs to include #define statements.

Your posting is not entirely clear, but I believe what you want is a CWEB source that will under CTANGLE produce a .h file and nothing else. The easiest way to do this is to invoke ctangle with a third file name argument; supposing you call the source file global.w and want to produce global.h, say

```
ctangle global - global.h
```

then the main output will be written to global.h instead of to global.c. The `-' is to indicate that you are not specifying a change file (without it, global.h would be taken to be the change file name).

If on the other hand you write a source file containing `@(global.h @>=' but no unnamed modules, and call CTANGLE normally, it will

still produce a main (.c) output file, which only contains something if you have used `@d' somewhere but not `@h' (the latter would redirect the preprocessor definitions to the auxiliary (.h) output file). I would call this a bug; in the function |phase_two| of CTANGLE where writing output is controlled, all the information is available to avoid writing the main output file if there is not going to be anything in it. The fact is however that CTANGLE opens its output file very early, even before opening its input file, and there is no undoing this. A warning is in place in here: suppose you have a partially literate project, and one of the source files is an ordinary C file without CWEB source, say `illit.c'; then if on some day you accidentally give the command `ctangle illit', the CTANGLE will complain that it can't find `illit.web' AFTER having replaced `illit.c' by an empty file, so you may hope to have made a backup of it... By the way, CWEAVE has a similar behaviour, except that the output file will not be empty, but rather contains "\input cwebma". Needless to say, CWEBx has a more decent behaviour in this respect.

From: **Roland Kaufmann**
Date: 04 Nov 1994

Ender Olcayto seems to want to tangle into a single .h file. I have been using a strategy where one huge (well, for my taste.. :-) web, which produces several .c and .h files using CWEBs @(file @> mechanism and it worked just fine. I didn't encounter the difficulties of producing a spurious .c file because my WEB tangled into several .c files anyway and incidentally the names of the .h and .c files matched. (See my earlier posting for more details).

From: **Dave Hamilton**
Date: 04 Nov 1994

I use a DOS-ported version of CWEB and when I want to generate a cplusplus file from my web with an extension of .cpp rather than the default .c, I use the command 'ctangle file.w - file.cpp'. It seems to me that you could do something similar to generate only a header file.

WEB TeX to MWeb TeX to OWeb TeX

From: **Jeffrey McArthur**
Date: 11 Nov 1994

The company I work for is considering a major undertaking. We would like to create a multi-threaded version of TeX. The approach we would like to take is to convert the WEB (Pascal) version of TeX into a MWEB (Modula-2) of TeX, and from there move to OWEB (Oberon or Oberon-2 WEB which currently does not yet exist). To do this we would need programmers who are well versed in WEB, Pascal, Modula-2 and Oberon. The first target platform would be OS/2. To completely control development it may be necessary to implement our own version of an Oberon (or Oberon-2) compiler. Knowledge of compiler construction and OS/2 are also needed. We need to know if we can find people who could undertake this challanging task. If you would be interested in working on this project I would like to hear from you.

From: **Lee Wittenberg**
Date: 11 Nov 1994

The company I work for is considering a major undertaking. We would like to create a multi-threaded version of TeX. The approach we would like to take is to convert the WEB (Pascal) version of TeX into a MWEB (Modula-2) of TeX, and from there move to OWEB (Oberon or Oberon-2 WEB which currently does not yet exist). To do this we would need programmers who are well versed in WEB, Pascal, Modula-2 and Oberon. The first target platform would be OS/2. To completely control development it may be necessary to implement our own version of an Oberon (or Oberon-2) compiler. Knowledge of compiler construction and OS/2 are also needed. We need to know if we can find people who could undertake this challanging task. If you would be interested in working on this project I would like to hear from you.

I'm interested (intrigued might be a better word). I'm somewhat familiar with Oberon & compiler construction, but am completely ignorant about OS/2. On the other hand, I have quite a bit of experience with Spidery WEB, which should be useful in creating an OWEB system.

From: **Barry Schwartz**
Date: 11 Nov 1994

Jeffrey McArthur writes: The company I work for is considering a major undertaking. We would like to create a multi-threaded version of TeX. The approach we would like to take is to convert the WEB (Pascal) version of TeX into a MWEB (Modula-2) of TeX, and from there move to OWEB (Oberon or Oberon-2 WEB which currently does not yet exist).

Dare I suggest you use noweb, so you can use the same tool for whatever languages you finally wind up using?

From: **Jeffrey McArthur**
Date: 12 Nov 1994

Dare I suggest you use noweb, so you can use the same tool for whatever languages you finally wind up using?

TeX is currently written in Web. There is an existing MWeb. We currently have a version of TeX that compiles under TopSeed Pascal. We plan on converting TeX from Pascal Web to Modula-2 Web one procedure at a time. This allows us to use the Trip test to make sure that we don't break anything. While we do the port we use a sub-set of Modula-2 that will make it easy to port to Oberon. If things work as planned, the Modula-2 version will directly compile using an Oberon Compiler. Converting MWEB to an OWEB should be almost trivial (going from Pascal to Modula-2 is much more complex than going from Modula-2 to Oberon).

From: **Lee Wittenberg**
Date: 14 Nov 1994

Jeffrey McArthur writes, in response to the following question: "Dare I suggest you use noweb, so you can use the same tool for whatever languages you finally wind up using?" TeX is currently written in Web. There is an existing MWeb. We currently have a version of TeX that compiles under TopSeed Pascal. We plan on converting TeX from Pascal Web to Modula-2 Web one procedure at a time. This allows us to use the Trip test to make sure that we don't break anything. While we do the port we use a sub-set of Modula-2 that will make it easy to port to Oberon. If things work as planned, the Modula-2 version will directly compile using an Oberon Compiler. Converting MWEB to an OWEB should be almost trivial (going from Pascal to Modula-2 is much more complex than going from Modula-2 to Oberon).

Actually, going from WEB to noweb should also be trivial, and going from noweb (Modula) to noweb (Oberon) should be equally trivial. In fact, if you use the nocond filter, you can keep all three versions together in the same web, enabling you to make sure that the new language code in corresponding chunks actually accomplishes the same result as the original language chunk, thus increasing confidence in the correctness of the translation.

CWEAVE and MFC class wizard

From: **Dave Hamilton**
Date: 18 Nov 1994

This may seem like a strange combination of environments and tools to most readers of this newsgroup, but here goes... I am using CWEB under Windows NT and developing with Visual C++ and the Microsoft foundation class library. The environment includes a component called the "Class Wizard" that automates connecting class member functions to message ids. It does this by managing parts of a source file that are bracketed by the following two lines:

```
//{{AFX_MSG(Classname)
...
//}}AFX_MSG
```

I can tell the class wizard to look at my .w file instead of my .h and .cpp files for these special managed sections. The problem is that CWEB complains about the braces as part of comments. The class wizard requires that nothing precedes the "//" on the special comment lines (although it doesn't mind if there is something after it). This prevents me from using @q or @= to get CWEB to ignore the lines because anything between @q and @> or @= and @> must be all on one line. Any advice?

From: **Marc van Leeuwen**
Date: 21 Nov 1994

That a nasty little problem you have there, but you might have a few options. First of all why is there a problem? Because CWEAVE wants braces to be balanced in comments. This is really a relic from the Pascal origins of CWEAVE, since there comments are delimited by braces, and brace-counting is inevitable. So you could safely remove the code that does this from CWEAVE, and there would be no complaint any more (if you are using CWEB 3.2, you must not forget to invent a new name for the resulting system though, see the copyright notice:-); nevertheless your problem would reappear because the comment is converted to an argument to a formatting macro for comments (\SHC in your case), and TeX doesn't like (more exactly, doesn't recognise) macro arguments that

start with closing braces. I can think of some solutions that might work. I have focussed on the second line, since it appears that the first one is simply solved by adding `}}' at the end of it.

1. Tell the "Class Wizard" to look at the output from CTANGLE rather than at the .w file (so your .cpp or .h file). Then force an appripriate comment into that file by saying (in the second case) @=//}}AFX_MSG@> at the end of the line (CTANGLE respects line ends, so it will stay there). You might want to insert @q{{@> in front to keep other brace counters (e.g., your editor) happy.

2. Do as you have done until now (having the "Class Wizard" scan the .w file), and ignore the error messages from CWEAVE. It will remove/insert braces as it finds appropriate, and TeX will be able to process the output of CWEAVE.

3. Keep the brace-counting code in CWEAVE, but remove the error messages. This is institutionalising solution of 2.

4. Remove the brace-counting code from CWEAVE altogether, and insert other comments, like /* \gobble{{this vanishes! */ //}}AFX_MSG where you define \def\gobble#1{}. In the TeX file this would result in \C{ \gobble{{this vanishes! }\SHC{}}AFX_MSG } which TeX will interpret as a call of \C with ` \gobble...AFX_MSG ' as argument, which will ``expand'' to ` AFX_MSG '.

Proposal: OpenDoc/OLE literate programming tool

From: **Tony Coates**
Date: 28 Nov 1994

In the past I've written suggesting that I expected that one day a `Windows-style' literate programming application would supercede that text/TeX-based tools that exist now. While I am currently not so unhappy using FunnelWeb/LaTeX (or FunnelWeb/HTML), there are times I would love to be able to just start up a drawing program to include a diagram into my documentation, for example. Yes, there are ways of doing this with TeX, of course, but it is a more involved process than I would like.

This has led me to wonder recently whether OpenDoc and/or OLE, which are meant to allow one program to call the functionality of another, might be used to create a system in which the literate programming tool takes the form of a layer which extracts source code from the documentation/source file (be it a word-processor file or a (La)TeX file) and passes the code, suitable reconstructed, to a compiler or make utility or whatever. The idea would be to try and make this layer as generic as possible, or maybe have 3 layers: 1: doc -> litprog, 2: litprog processing, 3: litprog -> compiler, etc.

Does anyone have any thoughts on this? I would be interested to know which functions people don't have now or fear they would lose with such a proposal. For my own part, I am most interested in generating documentation which is viewed electronically, preferably with hypertext links, though I am aware that others prefer to view the documentation on paper, which leads to different needs, though I would expect that creating a tool to properly exploit both shouldn't be so hard, as with the LaTeX/HTML generation in noweb (and in my experimental version of FunnelWeb, for anyone who wants to beta-test). Anyway, I look forward to any thoughts the group may have.

From: **Jacob Nielsen**
Date: 29 Nov 1994

Tony Coates writes: In the past I've written suggesting that I expected that one day a `Windows-style' literate programming application would supercede that text/TeX-based tools that exist now. [...]

I see one major stopping block: not everyone uses Windows and it's my impression that this is especially true in this forum. No doubt that if literate-programming is to reach a wide audience, it is essential that we can do literate-programming in a windowing system with a real :-) word processing program.

This has led me to wonder recently whether OpenDoc and/or OLE, which are meant to allow one program to call the functionality of another, might be used to create a system in which the literate programming tool takes the form of a layer which extracts source code from the documentation/source file (be it a word-processor file or a (La)TeX file) and passes the code, suitable reconstructed, to a compiler or make utility or whatever. The idea would be to try and make this layer as generic as possible, or maybe have 3 layers: 1: doc -> litprog, 2: litprog processing, 3: litprog -> compiler, etc.

Does anyone have any thoughts on this? I would be interested to know which functions people don't have now or fear they would lose with such a proposal. [...]

What about the links/references, who handles them? In my opinion this has to be done by the computer.

What I really want is not only an automatic extraction of the code; I want to have the two standard things of nuweb, noweb etc. and the

joy of WYSIWYG: 1. Interleaving of documentation + code, and 2. Automatic indexing/crossreferencing of code. The documentation part (1.) is quite well served by any standard word processor today. It is because of my laziness (2.) that I do not write my literate programs in a word processor today. It would be great if OpenDoc/OLE/any standard could do these things, but I'd still stay with NEXTSTEP as my computing enviroment and I guess others wouldn't trade their Macs, X-terminals etc. for Windows.

Staying with the notion of an OpenDoc standard, perhaps we should think of a literate program not as documentation interleaved with code, but as documentation interleaved with code and indexing info; this way we could let the literate programming tool handle the indexing. I envision something like this: (the ``..." signals the different programs)

```
``Word processor''
This is just a little test.
``Litprog Code Editor''
<File: "test.cc">=
#include "todo.h"

int main() <Index: "main()">
{
doSomeThingUsefull();<Index: "doSomeThingUsefull()">
return 1;
}
``Word processor''
And the text continues.
```

This would look like this in the word processor:

```
This is just a little test.
"test.cc"  1 =
#include "todo.h"

int main()
{
doSomeThingUsefull();
return 1;
}
And the text continues.
```

with all the indexing of user defined names etc. is controlled by the Litprog Code Editor. Disclaimer: I know nothing about OLE/OpenDoc, so I can't say if this is possible. But then, who ever said a man can't dream?

Usage of CWEB

From: **Yuval Peduel**
Date: 30 Nov 1994

I've long been an advocate of literate programming, but until recently had no choice but to do it with nothing more than commenting conventions using an editor. Then I got the chance to look into the WEB family of tools and grabbed it. Unfortunately, the beautiful bubble seems to have burst. My attempts to use CWEB have resulted in completely incomprehensible output:

1. When I write code, I do not always go top-down. When creating a loop, I'll often design the loop invariant, the inner code to implement it, and then the loop primitive and the initialization code. But when I tried to set up a scrap (?) with: @<process one unit@>= before writing the loop which refers to @<process one unit@>, the cweave output came out all wrong: the curly brackets for the if statements ended up adjacent, on the line with the "if" and the Boolean expression, rather than bracketing the conditional code.

2. Under certain situations that I do not yet understand, line breaks that are clearly needed are not put in. In one case, five separate input lines, each a separate statement, were output by cweave on one line. More than once, several statements from inside a compound statement were put on one output line while the remaining statements from the same compound statement were not. In other cases, the "if", the Boolean expression, and part of the conditional code were merged with a preceding assignment statement while the else clause was handled perfectly.

I've also had minor but still annoying problems:

1. I like to organize my code along the LISP COND style, which translates into a C else-if chain. Cweave seems to insist on indenting further at each link of the chain.

2. I take full advantage of the Unix file system to give readable names to my files, using underscores to separate words. When cweave

takes the file name for use in the header, it can't handle this.

3. I think indentation is a great aid to readability, but two spaces per level is too small. I'd prefer 3 or 4, yet do not know how to do this.

While I can't say much about these minor problems, I do know that in some sense, the major problems are "my fault". To bring up CWEB, I FTPed the Tex, Mfont, CWeb, etc. sources and did a full install starting with config at every step along the way. When I was done, I ran all the examples in the cweb/examples directory and the results looked good. Thus the installation and the software seem O.K.

On the other hand, I patterned my code after what I saw in the examples, with what seemed to be only minor changes to accommodate my style. Surely I've made mistakes, but they can't be major and I still have no idea where or what they are. I chose CWEB rather than one of the language independent tools because I knew I would be programming in C and I thought that a language specific tool would provide more robust error recognition and handling.

I suppose my questions are:

1. Is my experience unique or do others experience this kind of tool fragility? Are these problems unique to CWEB or would I have similar problems with the other tools? (I.e. is the problem rooted in the C syntax, the macro processing limitations of Tex, a by-product of the CWEB implementation, or inherent in any attempt at a powerful text-based tool of this nature?)

2. Is there documentation to help newcomers? I've read the literate-programming FAQ, but seen none specific to CWEB. I have a copy of Knuth and Levy, but its usage information is minimal. Reading the code to find out how to use it is exactly what I'm trying to get away from.

3. Are there any additional tools that would make it easier to satisfy the finicky requirements of cweave. For example, since the C compiler tends to be finicky, I use indent to make sure that I haven't left out closing braces, etc., but indent isn't really applicable to CWEB files. Are there any useful Emacs modes or the like?

4. Where do I go from here?

I'm looking forward to getting both specific answers and to reading whatever general discussion these questions might generate.

From: **Wheeler Ruml**
Date: 01 Dec 1994

I tried to use CWEB two years ago, and was also frustrated by similar problems to the ones you describe. I'm in the middle of trying again, this time for C++ instead of C, and I'm just about to give up. As far as I can tell, the technology just isn't "out-of-the-box" yet. I'm having a terrible time getting my Makefile to work properly (GNU Make), and I miss the highly-developed C and C++ modes in Emacs.

Some of the formatting problems can be fixed using @; and such - read the CWEB docs carefully! Hacking the actually grammar is really hairy - check out some of the alternate CWEB variants, they have a cleaner layout. If anyone can offer some help with these sorts of configuration issues, I'd be grateful. I'm not willing to spend this much time fiddling with my tools and down-grading my expectations.

From: **Jacob Nielsen**
Date: 01 Dec 1994

Yuval Peduel writes: When I write code, I do not always go top-down. When creating a loop, I'll often design the loop invariant, the inner code to implement it, and then the loop primitive and the initialization code. But when I tried to set up a scrap (?) with: @<process one unit@>= before writing the loop which refers to @<process one unit@>, the cweave output came out all wrong: the curly brackets for the if statements ended up adjacent, on the line with the "if" and the Boolean expression, rather than bracketing the conditional code.

```
If I understand correctly, cweave produces:

    if (<Condition>) {
      <Contitional code>
    }

and you want:

    if (<Condition>)
    {
      <Contitional code>
    }
```

Welcome to the world of pretty-printing, style a la Knuth and Levy.

1. Is my experience unique or do others experience this kind of tool fragility? Are these problems unique to CWEB or would I have similar problems with the other tools? (I.e. is the problem rooted in the C syntax, the macro processing limitations of Tex, a by-product of the CWEB implementation, or inherent in any attempt at a powerful text-based tool of this nature?)

The rearranging of statements (if clauses etc.) are almost inevitable if you use any literate programming tools that does serious source code formatting. The problem arises when the programmer uses one convention for how the code should look and the tool uses another. If all adhered to the Knuth/Levy style of formatting code, there would be no problems. If you want "poor mans pretty-printing" you should take a look at noweb. noweb is independent of the programming language, but "poor mans pretty-printing" has been added for C. My definition of "poor mans pretty-printing": It typesets keywords in bold etc. but respects newlines, indentation, spaces and such.

Are there any useful Emacs modes or the like?

I think there is a CWEB-mode (web-mode?)

PS: I think that pretty-printed code looks good, but it seems to be too much trouble.

From: **Joachim Schrod**
Date: 01 Dec 1994

Are there any useful Emacs modes or the like?

Jacob Nielsen writes: I think there is a CWEB-mode (web-mode?)

IMNSHO: forget it. If you're used cc-mode and auctex, you'll throw it out immediately. In addition, it globally rebinds standard keys. Similar as web-mode does. For me, that's always a sign that authors did not understand Emacs concepts. There exists no really good Emacs support for literate programming until now, and that's a bad sign, actually. (I have to admit that I don't like nuweb-mode, either. To edit the source in an own window defeats the whole purpose of literate programming: Handling documentation and source as *one* unit.)

PS: I think that pretty-printed code looks good, but it seems to be too much trouble.

Me too, and the success of cc-mode/font-lock/hilit (or Borland IDE-style editors, for that matter) shows that people like it. The problem is IMO more the current fixed formatting engine of CWEB than the process of pretty-printing as a whole.

From: **Marc van Leeuwen**
Date: 01 Dec 1994

Yuval Peduel writes: I've long been an advocate of literate programming, but until recently had no choice but to do it with nothing more than commenting conventions using an editor. Then I got the chance to look into the WEB family of tools and grabbed it. Unfortunately, the beautiful bubble seems to have burst.

From what follows I gather that your problems are partly that you are dissatisfied with the conventions (and a bug) built into Levy/Knuth CWEB, and partly that you are experiencing parsing problems that CWEAVE fails to diagnose, but which completely screw up your output. I would urge you to try my CWEBx system, that was designed to be more flexible, informative, comprehensible, and correct than CWEB, yet basically works just like it. In fact, when using compatibility mode (command line option `+c') you should be able to process Levy/Knuth CWEB sources just as they are, but hopefully with more pleasant output. Limitations: the current version is a beta version; only fairly basic C++ is supported; the manual is in the process of being rewritten and therefore not complete (but the recent additions are listed in a separate file). Below I will indicate how the possibilities of CWEBx relate to your problems.

My attempts to use CWEB have resulted in completely incomprehensible output: 1. When I write code, I do not always go top-down. When creating a loop, I'll often design the loop invariant, the inner code to implement it, and then the loop primitive and the initialization code. But when I tried to set up a scrap (?) with: @<process one unit@>= before writing the loop which refers to @<process one unit@>, the CWEAVE output came out all wrong: the curly brackets for the if statements ended up adjacent, on the line with the "if" and the Boolean expression, rather than bracketing the conditional code.

So you dislike the ugly brace style promoted by K&R, which Levy/Knuth CWEB has implemented. CWEBx gives you the choice between three brace styles, two of which (including the default) align opening and closing braces vertically.

2. Under certain situations that I do not yet understand, line breaks that are clearly needed are not put in. In one case, five separate input lines, each a separate statement, were output by CWEAVE on one line. More than once, several statements from inside a compound statement were put on one output line while the remaining statements from the same compound statement were not. In other cases, the "if", the Boolean expression, and part of the conditional code were merged with a preceding assignment statement while the else clause was handled perfectly.

You have definitely run into syntax problems here, which may have a number of causes. Common ones are macro invocations that are used in a way other than as an expression (which is what they look like), such as a complete statement (no semicolon after it), and typedef identifiers that CWEB does not know about; the former problem can be solved using `@;', the latter using `@f' or `@s' (or in CWEBx outside compatibility mode, even better by using the `@h' command to specify that included header files should be scanned for typedef declarations). To diagnose your problem, you may like to view any irreducible scrap sequences (which is a technical term for what remains from input that could not be completely digested by the parser). To obtain this, place `@1' in your first section, or for CWEBx specify a `+d' command option to CWEAVE.

I've also had minor but still annoying problems: 1. I like to organize my code along the LISP COND style, which translates into a C else-if chain. CWEAVE seems to insist on indenting further at each link of the chain.

This is strange; CWEB certainly does process `if.. else if ... else if' chains without increasing indentation. Maybe you have the same problems as under 2. above here?

2. I take full advantage of the Unix file system to give readable names to my files, using underscores to separate words. When CWEAVE takes the file name for use in the header, it can't handle this.

This is a bug in Levy/Knuth CWEB. CWEBx handles special characters in file names correctly.

3. I think indentation is a great aid to readability, but two spaces per level is too small. I'd prefer 3 or 4, yet do not know how to do this.

In Levy/Knuth CWEB indentation is fixed to one `em' (the width of a \quad). In CWEBx you could say \indentation{2em} in limbo, or \indentation{1cm}, or whatever unit of indentation you like. This is really not a property of the CWEB programs, but of the TeX macro format used.

I chose CWEB rather than one of the language independent tools because I knew I would be programming in C and I thought that a language specific tool would provide more robust error recognition and handling.

It surely should.

I suppose my questions are: 1. Is my experience unique or do others experience this kind of tool fragility? Are these problems unique to CWEB or would I have similar problems with the other tools? (I.e. is the problem rooted in the C syntax, the macro processing limitations of Tex, a by-product of the CWEB implementation, or inherent in any attempt at a powerful text-based tool of this nature?)

I would say all your problems can be solved within the CWEB context, and most are solved in CWEBx. There are a few fundamental problems, but you are not very likely to run into them. (One is for instance typedef declarations that are local to a block; CWEAVE has no idea of lexical ranges (which might be quite disconnected in the CWEB source) and simply assumes all typedef declarations to be global. This could be a problem in C++, particularly when using templates, but for C I have never seen a local typedef.)

2. Is there documentation to help newcomers? I've read the literate-programming FAQ, but seen none specific to CWEB. I have a copy of Knuth and Levy, but its usage information is minimal. Reading the code to find out how to use it is exactly what I'm trying to get away from.

CWEBx comes with a manual that tries to explain all relevant issues in a much more elaborate way than the Levy/Knuth manual.

3. Are there any additional tools that would make it easier to satisfy the finicky requirements of CWEAVE. For example, since the C compiler tends to be finicky, I use indent to make sure that I haven't left out closing braces, etc., but indent isn't really applicable to CWEB files. Are there any useful Emacs modes or the like?

CWEBx's CTANGLE counts braces and parentheses in every macro or module body, and reports and "corrects" any ones that are unbalanced. I think this is easier and more reliable than any brace matching done by an editor (they tend to get confused by the mixture of different lexical conventions that is used in CWEB source code).

From: **Marc van Leeuwen**
Date: 01 Dec 1994

Wheeler Ruml writes: I tried to use CWEB two years ago, and was also frustrated by similar problems to the ones you describe. I'm in the middle of trying again, this time for C++ instead of C, and I'm just about to give up. As far as I can tell, the technology just isn't "out-of-the-box" yet. I'm having a terrible time getting my Makefile to work properly (GNU Make), and I miss the highly-developed C and C++ modes in Emacs.

What's the problem with make files? The only one I know of is that if CTANGLE writes multiple files (e.g., a program file and a header file) then these will always be updated when any change is made to that source file. You could solve this by moving the old files into a subdirectory before running CTANGLE, and afterwards compare the new files with the old ones, moving the old files back to replace the new ones if they are equal, or removing them if not. All this could be specified in the make file. This is like noweb's cpif script, except that it does not require files to be written on stdout. The main problem I foresee is that files may change due to changing #line directives, while their actual contents is constant. For program files I would prefer to recompile the file if any #line directives have changed, lest my debugger would get confused, but for header files I might prefer to keep the old (incorrect) #line directives in order to preserve the older timestamp on the file; it is not difficult to write a comparison program that ignores lines starting with #line.

From: **Balasubramanian Narasimhan**
Date: 01 Dec 1994

Yuval Peduel writes: I've long been an advocate of literate programming, but until recently had no choice but to do it with nothing more than commenting conventions using an editor. Then I got the chance to look into the WEB family of tools and grabbed it. Unfortunately, the beautiful bubble seems to have burst. My attempts to use CWEB have resulted in completely incomprehensible output:

My attempts at literate programming in CWEB have left me disheartened too. When I started work on the Diehard tests for Random Number Generators, I thought here was a project that would really benefit from a literate programming style. However, a few attempts mangled both the typesetting and code and I had to abandon the effort. I have one example ready at hand so users can duplicate one problem. (This example below was a first attempt some time ago.) Run the file below thru' cweave and TeX. Things should be nice. Now change every instance of the word "class" into "CLASS" and run it thru' cweave and TeX and see how it is typeset.

```
***************Begin example.w*********************
% Rngs: A package of Random Number Generators by George~Marsaglia,
%       B.~Narasimhan, and Arif~Zaman.

%\nocon % omit table of contents
\datethis % print date on listing
\def\DH{Diehard}
\def\TCL{Tcl/Tk}
\def\RNG{RNG}
\def\RNGS{RNGs}

@** Introduction. This document forms part of the \DH{} package,
written by George~Marsaglia, B.~Narasimhan, and Arif~Zaman. It
describes the components of the header file which contains
implementation limits as well as function prototypes. Anyone who
wishes to add/or modify \RNGS{} should include this file.

@ Here is an overview of the organization. The entire header file is
enclosed within a conditional macro to prevent the definitions being
invoked repeatedly---the definitions obtain only when the variable
|_RNG_H| is undefined, which is the case during the first include.
This is so standard a technique that we shall not discuss
such little tricks henceforth.

@c
```

```
#ifndef _RNG_H
@<Constant definitions@>@/
@<Variable type definitions@>@/
@<Header files to include@>@/
@<Function prototypes@>@/
@<Other useful definitions@>@/
#define _RNG_H
#endif
```

@ We need some constants that define the limits of our programs. These constants can be changed if necessary. The |MAX_RNGS| constant indicates the maximum number of random number generators that can be added. In addition, the |CLASS| allows us to use a single header file with external functions correctly visible.

```
@<Constant definitions@>=
#ifdef _DH_MAIN
#define CLASS
#else
#define CLASS extern
#endif
#define MAX_RNGS 50
```

@ We need three new variable types: one for keeping track of the values returned by a random number generator, another for storing the value of the generator, and finally, a structure that can be used to store the state of a {\sl generic\/} random number generator.

```
@<Variable type...@>=
@<Typedef for value returned by rng@>@/
@<Typedef for mixed value@>@/
@<Structure for storing state of an rng@>
```

@ Some random number generators return sequences of reals while others return sequences of integers. (Throughout this document, when we refer to integers or unsigned integers, or doubles, we mean long integers, unsigned long integers, and doubles as defined in the language C.) The |rng_type| definition declares three kinds of values that might be returned by \RNGS: |RNG_DOUBLE| for a double real value, |RNG_ULONG| for an unsigned long integer, and |RNG_ILONG| denotes a long integer. It is up to us to keep track of the type of the value returned by any generator.

```
@<Typedef for value returned by rng@>=
typedef enum {RNG_DOUBLE, RNG_ULONG, RNG_ILONG} rng_type;
```

@ We need a variable to store the value returned by any \RNG. The following type definition is natural.

```
@<Typedef for mixed value@>=
union mixed_value {
    unsigned long uival;
    long int ival;
    double dval;
  } mixed_value;
```

@ Next, we need a structure for storing the state of {\sl any\/} \RNG. More complicated \RNGS{} usually use a table of values for generating the next number in a sequence. The tables might be made up of real numbers or signed integers or unsigned integers. Typically, the table tends to be homogeneous, i.e., it is either composed of reals, or integers but not a mixture of both. We need some more constants.

```
@<Constant...@>=
#define MAX_TBL_SIZE 1024
#define MAX_INDICES 8
#define MAX_NAME_LEN 25
```

@ Now on to a structure for storing the state and related information

pertaining to an \RNG. Some fields suggest themselves. Every \RNG{} will be uniquely identified by an index stored in |index|. The string |name| of maximum length |MAX_NAME_LEN| will hold a descriptive name of an \RNG, |type|, the type of value the generator returns, |bits| the number of valid bits in the generator, |start| and |end|, the starting and ending bits, if applicable. The last two fields are expected to be zero when not applicable. The table itself is |table| and |table_type| reveals what type of values are stored in the table. Indices into |table| are almost surely necessary and |table_inds| provides up to a maximum of |MAX_INDICES|. We shall be generous and allow for another similar table |x_table| analogous to |table| for other unseen possibilities---it is left to the programmer to use them as he sees fit.

```
@<Structure...@>=
typedef struct rng_info
{
  int index;               /* The index of the generator. */
  char name[MAX_NAME_LEN]; /* Name of the Rng. */
  rng_type type;           /* Type value Rng returns. */
  int bits;                /* No. of valid bits in the rng. */
  int start;               /* Start of kosher bits. */
  int end;                 /* End of kosher bits. */
  rng_type table_type;     /* Type of value in table. */
  mixed_value table[MAX_TBL_SIZE]; /* The table itself. */
  int table_inds[MAX_INDICES]; /* Indices into table. */
  rng_type x_table_type;     /* Type of value in the extra table. */
  mixed_value x_table[MAX_TBL_SIZE]; /* The extra table. */
  int x_table_inds[MAX_INDICES]; /* Indices into extra table. */
} rng_info;
```

@ We must include the standard \TCL, math and string header files since we will be using \TCL, math and string functions.

```
@<Header files...@>=
#include <tcl.h>
#include <math.h>
#include <string.h>
```

@ We now define our function prototypes. These can be roughly divided as follows.

```
@<Function prototypes@>=
@<Tcl oriented function prototypes@>@/
@<Diehard function prototypes@>@/
@<Random Number Generator function prototypes@>@/
```

@ The \DH{} package uses \TCL, and therefore some TCLish conventions need to addressed. \TCL{} passes a handle to a Tcl intepreter that can be used for communication. It seems to be a waste to pass the interpreter handle everytime and so we shall use two functions for accessing and storing interpreter handles. Note that this arrangement imposes limitations on this package---most notably, support for multiple interpreters is lost. We might address this deficiency at a later point.

```
@<Tcl oriented...@>=
extern void set_interpreter(Tcl_Interp *a);
extern Tcl_Interp *get_interpreter(void);
```

@ The following functions are defined elsewhere in respective files. However, they are so crucial to our design that we provide a brief description of each right here. The functions |get_rng_index| and |set_rng_index| are the accessor and modifier functions for a global variable that holds the index of the currently-chosen \RNG.

```
@<Diehard function...@>=
```

```
CLASS int get_rng_index(void);
CLASS void set_rng_index(int a);

@ The function |set_current_rng| allows us to set the current \RNG{}
to any one of the available \RNGS. Note that it takes a pointer to
an \RNG{} function as its argument. The function |current_rng|
returns a value from the currently chosen \RNG. The value returned is
a pointer to a variable of type |mixed_value|.

@<Diehard function...@>=
CLASS void set_current_rng(mixed_value* (*a)());
CLASS mixed_value *(*current_rng)(void);

@ The function |get_rng_info| returns a pointer to a structure of type
|rng_info|. This structure should always be current, and anytime an
\RNG{} is chosen, one must ensure that this structure contains
pertinent information as well as a snapshot of the state of the
\RNG. The additional functions |get_rng_type|, |set_rng_type|,
|get_rng_bits|, |set_rng_bits|, |get_rng_name|, |set_rng_name| are
provided for conveniently accessing and modifying commonly used fields
of the |rng_info| structure.

@<Diehard function...@>=
CLASS rng_info *get_rng_info(void);
CLASS rng_type get_rng_type(void);
CLASS void set_rng_type(rng_type a);
CLASS int get_rng_bits(void);
CLASS void set_rng_bits(int a);
CLASS char *get_rng_name(void);
CLASS void set_rng_name(char *a);

@ The boolean function |rng_properly_chosen| can be used to avoid
errors---it returns |true| when all is well.

@<Diehard function...@>=
CLASS int rng_properly_chosen(void);

@ This section lists all the \RNGS{} implemented. We provide three
random number generators at present: Lagged-Fibonacci, KISS, and
Super~Duper.

@<Random...@>=
#ifdef _DH_MAIN
@<Super Duper@>@/
@<Lagged Fibonacci@>@/
@<KISS@>@/
#endif

@ This section explains how \RNGS{} should be designed for use with
\DH. For every \RNG{}, there must be at least four
routines: (a)~an initializing routine, that seeds the state of the
\RNG{} possibly based on user chosen seed values, (b)~a routine that
computes the next random number in the sequence and returns a pointer
to a |mixed_value| type, (c)~a routine that stores the state of the
\RNG{} in a structure of type |rng_info|, a pointer to which is passed
to the routine, and (d)~a routine that sets the state of the \RNG{}
from a structure of type |rng_info|, a pointer to which is again
passed to the routine. These requirements are best illustrated by a
detailed example, say Marsaglia's Super~Duper.

@<Super Duper@>=
@<Super-Duper initializer prototype@>@/
@<Super-Duper generator prototype@>@/
@<Super-Duper state saver prototype@>@/
@<Super-Duper state setter prototype@>@/

@ The initialization routine for Super Duper is |supdup_init| and it takes
a pointer to |rng_info| as argument and returns an integer code. The
```

```
code will be either |TCL_OK|, indicating that all was well, or
|TCL_ERROR|, indicating something was amiss.

@<Super-Duper initializer prototype@>=
extern int supdup_init(rng_info *a);

@ The Super duper generator itself is |supdup| and it returns a
pointer to |mixed_value|.

@<Super-Duper generator prototype@>=
extern mixed_value *supdup(void);

@ The routine that will save the state of Super Duper in an |rng_info|
structure is |supdup_save_state|.

@<Super-Duper state saver prototype@>=
extern void supdup_save_state(rng_info *a);

@ And finally, |supdup_set_state| will set the state of Super Duper from
an |rng_info| structure.

@<Super-Duper state setter prototype@>=
extern void supdup_set_state(rng_info *a);

@ All other generators are similar with some minor differences. The
Lagged~Fibonacci generator, for example, has many specialized
sub-generators for efficiency. They are described in the file
|fibo.c|.

@<Lagged Fibonacci@>=
extern mixed_value *fibomulmod(void);
extern mixed_value *fiboplusmod(void);
extern mixed_value *fiboxormod(void);
extern mixed_value *fibosubmod(void);
extern mixed_value *fibomul32(void);
extern mixed_value *fiboplus32(void);
extern mixed_value *fiboxor32(void);
extern mixed_value *fibosub32(void);
extern int fibo_init(rng_info *a);
extern void fibo_save_state(rng_info *a);
extern void fibo_set_state(rng_info *a);

@ This section pertains to the KISS generator of Marsaglia and
Zaman. KISS stands for ``Keep It Simple, Stupid.''

@<KISS@>=
extern int kiss_init(rng_info *a);
extern mixed_value *kiss(void);
extern void kiss_save_state(rng_info *a);
extern void kiss_set_state(rng_info *a);

@ The following definitions are for convenience.

@<Other...@>=
#define ulong_rng_value (*(unsigned long *)(*current_rng)())
#define int_rng_value (*(int *)(*current_rng)())
#define double_rng_value (*(double *)(*current_rng)())
#define ulong_rng_value_ptr (unsigned long *)(*current_rng)()
#define ulong_rng_value_ptr (int *)(*current_rng)()
#define double_rng_value_ptr (double *)(*current_rng)()
#define max(x,y) ((x) > (y) ? (x) : (y))
#define lg(x) (log(x)/log(2.0))

@* Index.
Here is a list of the identifiers used, and where they appear. Underlined
entries indicate the place of definition. Error messages are also shown.
```

```
************End example.w*************************
```

From: **Andrew John Mauer**
Date: 01 Dec 1994

Yuval Peduel writes: 2. I take full advantage of the Unix file system to give readable names to my files, using underscores to separate words. When cweave takes the file name for use in the header, it can't handle this.

This is a minor issue that I have also run across in noweb. I found the simplest solution was to bend a little and use hyphens (`-') rather than underscores for separators in filenames. It makes life easier.

From: **Lee Wittenberg**
Date: 01 Dec 1994

Balasubramanian Narasimhan writes: My attempts at literate programming in CWEB have left me disheartened too. When I started work on the Diehard tests for Random Number Generators, I thought here was a project that would really benefit from a literate programming style. However, a few attempts mangled both the typesetting and code and I had to abandon the effort. I have one example ready at hand so users can duplicate one problem. (This example below was a first attempt some time ago.)

Try noweb. It's much simpler.

Run the file below thru' cweave and TeX. Things should be nice. Now change every instance of the word "class" into "CLASS" and run it thru' cweave and TeX and see how it is typeset.

I think you mean it the other way around. The program you submitted used `CLASS' rather than `class'. Since the latter is a reserved word in C++ (which is accepted by CWEB), it's not surprising that the typesetting is different. Given your definition of CLASS:

```
#ifdef _DH_MAIN
#define CLASS
#else
#define CLASS extern
#endif
```

You should probably have an "@f CLASS int" in the definitions part of the chunk (@s will do as well). This will work even when you change all the `CLASS's to `class'.

From: **Marc van Leeuwen**
Date: 02 Dec 1994

Balasubramanian Narasimhan writes: My attempts at literate programming in CWEB have left me disheartened too. When I started work on the Diehard tests for Random Number Generators, I thought here was a project that would really benefit from a literate programming style. However, a few attempts mangled both the typesetting and code and I had to abandon the effort. I have one example ready at hand so users can duplicate one problem. (This example below was a first attempt some time ago.) Run the file below thru' cweave and TeX. Things should be nice. Now change every instance of the word "class" into "CLASS" and run it thru' cweave and TeX and see how it is typeset.

Well, actually the example contained |CLASS| rather than |class|, but I got the point: with |class| it works, with |CLASS| it doesn't. Now you might be alerted by the fact that |class| is a keyword in C++, and (Levy/Knuth) CWEB handles C++ (no way to shut this off). Apart from this your program had a few weak spots:

```
#ifdef _DH_MAIN
#define CLASS
#else
#define CLASS extern
#endif
```

This is of course relevant to |CLASS|: it stands for the keyword |extern| (or for nothing). If you want to get proper formatting, you must inform CWEAVE that |CLASS| is not an ordinary identifier, but is used as a keyword. The proper way to say this is to specify "@f CLASS extern" in some section (e.g., the one containing the lines above) so that CWEAVE will treat |CLASS| just like it would treat |extern|. You cannot expect formatting to be proper if you are doing subtle things behind CWEAVE's back (and no, CWEAVE does not

attempt to expand macros; imagine the mess that would give in more complicated cases).

```
@<Typedef for mixed value@>=
union mixed_value {
     unsigned long uival;
     long int ival;
     double dval;
} mixed_value;
```

You said `typedef' in the module name, but didn't you forget to say it in the module itself? Better insert it there, or you C compiler will complain.

```
#include <tcl.h>
```

It appears that a typedef identifier |Tcl_Interp| is being defined in <tcl.h>. Then you should also tell this to CWEAVE: "@f Tcl_Interp int" so that CWEAVE will know to treat |Tcl_Interp| as a type. With these three small changes, your program comes out beautifully, whether using |class| or |CLASS| or something else.

So the really puzzling question is: why did it work well without the changes, provided you use |class|? Well, in your case you were in fact lucky that |class| is a C++ keyword, since this happened to make your program come through the parser, even though it is not proper C++ code (your use of |class| does not match the way it is used in C++). The grammar used by CWEAVE has some rules that are a bit too general, and they matched you use of |class| although they were really meant for different purposes. And what about |mixed_value|? Again there was C++ to your rescue, since for C++ a declaration of the form |union x { ... }| is treated as if it were |typedef union x { ... } x| in C; so your code actually was interpreted as if it were

```
typedef union mixed_value { ... } mixed_value;
mixed_value mixed_value;
```

in other words, |mixed_value| is both declared a typedef identifier, _and_ a variable of that type (compilers don't like this in one same scope, but CWEAVE is not that picky). So despite the fact that you forgot |typedef|, CWEAVE treated |mixed_value| as a typedef identifier, which is in fact what you had intended. The declarations involving |Tcl_Interp| did come out wrong, but the effect was not too dramatic, and you probably overlooked it.

Conclusion: if you want nice output, you should inform CWEAVE about anything that it needs to know about identifiers being used in unusual ways, and `@f' (or `@s') is often the way to do this. For types defined in header files, my version of CWEB, called CWEBx, will be able to do without `@f' lines, provided it can locate the header files in question.

From: **Balasubramanian Narasimhan**
Date: 02 Dec 1994

Marc van Leeuwen writes: <Lots of useful information deleted..>

I wish to thank Marc for his followup. Just a minor point. The missing typedef that Marc refers to was a result of my bungled editing. In any case, his analysis was an eye-opener.

From: **Yuval Peduel**
Date: 03 Dec 1994

First, my thanks to all who responded. Even the discouraging messages were helpful. On some of the specific points:

the cweave output came out all wrong: the curly brackets for the if statements ended up adjacent, on the line with the "if" and the Boolean expression, rather than bracketing the conditional code.

```
Jacob Nielsen wrote: If I understand correctly, cweave produces:
    if (<Condition>) {
      <Contitional code>
    }

and you want:

    if (<Condition>)
    {
      <Contitional code>
    }
```

My apologies for not making myself clear. What I actually got from cweave was something like:

```
     if (<Condition>) {  }
       <Conditional code>
```

and:

```
     for (;;) {  }
       <For body>
```

This is a somewhat more severe problem than one on pretty-printing style. :-(

Welcome to the world of pretty-printing, style a la Knuth and Levy.

I actually prefer having the open brace on the same line as the if, for, or while, so I can't complain about their choice of default. But I gather that this and such other parameters as the indentation level are not select-able by the user. This strikes as me reasonable in a package written for one person's use, but not for a general release.

The rearranging of statements (if clauses etc.) are almost inevitable if you use any literate programming tools that does serious source code formatting. The problem arises when the programmer uses one convention for how the code should look and the tool uses another. If all adhered to the Knuth/Levy style of formatting code, there would be no problems.

I understand and appreciate the need to do some code moving for full pretty-printing. I've also read about the advantages in comprehensibility that full formatting can provide. Nonetheless, I still see the need for some control over the process. After all, some of us have weaker eyes and need stronger clues, larger fonts, etc.

If you want "poor mans pretty-printing" you should take a look at noweb. noweb is independent of the programming language, but "poor mans pretty-printing" has been added for C. My definition of "poor mans pretty-printing": It typesets keywords in bold etc. but respects newlines, indentation, spaces and such.

If I get everything else working and the Knuth/Levy style becomes my primary problem, I'll consider this. In the meantime, I have bigger problems.

Marc van Leeuwen wrote: You have definitely run into syntax problems here, which may have a number of causes. Common ones are macro invocations that are used in a way other than as an expression (which is what they look like), such as a complete statement (no semicolon after it), and typedef identifiers that CWEB does not know about; the former problem can be solved using `@;', the latter using `@f' or `@s' (or in CWEBx outside compatibility mode, even better by using the `@h' command to specify that included header files should be scanned for typedef declarations). To diagnose your problem, you may like to view any irreducible scrap sequences (which is a technical term for what remains from input that could not be completely digested by the parser). To obtain this, place `@1' in your first section, or for CWEBx specify a `+d' command option to CWEAVE.

This is both good news and depressing. Good in that it gives me hope that there is a path out. Depressing in that one has to appeal to all of you out there to get this info.

I would say all your problems can be solved within the CWEB context, and most are solved in CWEBx. There are a few fundamental problems, but you are not very likely to run into them. (One is for instance typedef declarations that are local to a block; CWEAVE has no idea of lexical ranges (which might be quite disconnected in the CWEB source) and simply assumes all typedef declarations to be global. This could be a problem in C++, particularly when using templates, but for C I have never seen a local typedef.)

CWEBx comes with a manual that tries to explain all relevant issues in a much more elaborate way than the Levy/Knuth manual.

I'll be looking at CWEBx in general and this manual in particular. Thanks. So far, after reading the messages my original post brought out, I'd have to agree with Wheeler Ruml when he says:

the technology just isn't "out-of-the-box" yet

While I am willing to pursue it for a while longer, there is no way I can introduce CWEB as it stands for general use in my current environment. On the other hand, I still don't understand why this is the case. The individual pieces should all be well-understood by now and while putting them together is far from trivial, we are still dealing with a limited domain, so it should not be possible. What am I missing here? (If the response is, "look at the poor error handling of the C compilers out there, why should CWEB be any better?", I'd have to say "ouch" and then reposte with, "but we are concerned with the human interface; they aren't!")

From: **Tommy Marcus McGuire**
Date: 06 Dec 1994

Yuval Peduel wrote: My apologies for not making myself clear. What I actually got from cweave was something like:

```
     if (<Condition>) {  }
       <Conditional code>
```

and:

```
     for (;;) {  }
       <For body>
```

This is a somewhat more severe problem than one on pretty-printing style. :-(

You aren't kidding. Could you post some of the code scraps (sections, whatever) that produce this kind of result? It has been quite a while since I used CWEB, but I never saw anything like that unless you had the <Conditional code> line outside the braces.

From: **Yuval Peduel**
Date: 09 Dec 1994

Marc van Leeuwen writes: You have definitely run into syntax problems here, which may have a number of causes. Common ones are macro invocations that are used in a way other than as an expression (which is what they look like), such as a complete statement (no semicolon after it), and typedef identifiers that CWEB does not know about; the former problem can be solved using `@;', the latter using `@f' or `@s' (or in CWEBx outside compatibility mode, even better by using the `@h' command to specify that included header files should be scanned for typedef declarations). To diagnose your problem, you may like to view any irreducible scrap sequences (which is a technical term for what remains from input that could not be completely digested by the parser). To obtain this, place `@1' in your first section, or for CWEBx specify a `+d' command option to CWEAVE.

I have taken this advice, gone through my code, and fixed numerous problems. Some I identified just by reading the code, some by looking at the output of indent applied to individual fragments, and some by running CTANGLE. However, there are problems that I just cannot see. Here is an example of a short CWEB file that results in inappropriate code formatting. I'm sure the error is mine, but where is it?

```
\@*Test.
@1
This is an attempt to figure out what goes wrong with my output.

@c
void
foo(int bar)
{
   @<try reading the data@>;
}

@ This is the section that hasn't come out right.

@<try reading the data@>=
no_data_reads = 0;
for (;;) {
   bytes_read = (*port->foo.spd->spd_mbuf_read)(port->foo.handle,
                                                &mbuf_chain,
                                                FLAG,
                                                &error_byte,
                                                &error);
   if (error) {
```

```
      break;
   }
   else if (error_byte != SPD_RCURG) {
      break;
   }
   else break;
}
@ A termination section.
```

Any assistance welcomed. In response to my complaint about the handling of braces for conditional code (after if's, for's, etc.) Tommy McGuire writes:

You aren't kidding. Could you post some of the code scraps (sections, whatever) that produce this kind of result? It has been quite a while since I used CWEB, but I never saw anything like that unless you had the <Conditional code> line outside the braces.

Fortunately or unfortunately, I can't. After I went through the code fixing all the syntax errors I could find, this particular problem vanished. I still have no idea of which syntax errors caused which output problems.

I did, however, get the impression, which may be wrong, that both CWEB and CWEBx are very perturbed by a section in limbo: a section whose name is not referenced. (Just going through the code to make sure that every named fragment was referenced before it was defined seemed to significantly improve the output.) I can understand why this might be a problem for CTANGLE, but why should CWEAVE care?

From: **Marc van Leeuwen**
Date: 12 Dec 1994

Yuval Peduel writes: However, there are problems that I just cannot see. Here is an example of a short CWEB file that results in inappropriate code formatting. I'm sure the error is mine, but where is it? [...]

```
    bytes_read = (*port->foo.spd->spd_mbuf_read)(port->foo.handle,
                                                 &mbuf_chain,
                                                 FLAG,
                                                 &error_byte,
                                                 &error);
    if (error) {
       break;
    }
```

The "error" is the identifier |error|, which is indeed yours. The problem is that CWEAVE treats |error| as a reserved word, mainly so that it will come out in boldface when you write an `#error' preprocessor directive. Although the identifiers that can follow `#' in a preprocessor line are not all reserved words in C or C++, CWEAVE will still treat them like that. A solution to this problem in your case is to insert a line `@f error x', to demote |error| to an ordinary identifier. Note that there are other identifiers of this kind that are likely to cause trouble, for instance |line| and |undef|. I have always found this behaviour of CWEAVE irritating, and have recently changed CWEBx so that it will treat identifiers after `#' specially _only_ in that context, so your example causes no problem in CWEBx. A complete list of all reserved words appears in section 28 of the Levy/Knuth CWEAVE listing (note that all C++ keywords are there; to use them as identifiers in C requires similar precautions as for |error|). For CWEBx they are in section 117, but you needn't look it up; the only anomaly is |va_dcl| which is there for historic reasons (types that are defined in certain ANSI header files, like |FILE|, are also predefined in CWEAVE, whether or not you include that header file).

I did, however, get the impression, which may be wrong, that both CWEB and CWEBx are very perturbed by a section in limbo: a section whose name is not referenced. (Just going through the code to make sure that every named fragment was referenced before it was defined seemed to significantly improve the output.) I can understand why this might be a problem for CTANGLE, but why should CWEAVE care?

I don't understand this. First of all a "section in limbo" is a contradiction in terms, since limbo is the TeX text before the first section. I assume you meant a section that is defined (or cited) but never used. This causes a warning message by CWEAVE both in CWEB and CWEBx, since it could indicate an oversight or typing error on the part of the programmer, but it should not otherwise affect the output of CWEAVE. And CTANGLE doesn't care about unreferenced modules at all, although it will complain about undefined ones. Certainly it should make no difference whether a module is defined before it is used or the other way around: both are perfectly valid (although the former is a bit less customary), and should lead to well-formatted output.

From: **Yuval Peduel**

Date: 12 Dec 1994

Marc van Leeuwen writes: The "error" is the identifier |error|, which is indeed yours. The problem is that CWEAVE treats |error| as a reserved word, mainly so that it will come out in boldface when you write an `#error' preprocessor directive. Although the identifiers that can follow `#' in a preprocessor line are not all reserved words in C or C++, CWEAVE will still treat them like that. A solution to this problem in your case is to insert a line `@f error x', to demote |error| to an ordinary identifier.

Thank you. Now that I understand this, it seems crystal clear, though a bit perverted. I would never have found this on my own.

I have always found this behaviour of CWEAVE irritating, and have recently changed CWEBx so that it will treat identifiers after `#' specially _only_ in that context, so your example causes no problem in CWEBx.

True. I did download CWEBx, read the manual, and try it on my program. The results were not identical to CWEB, but they seemed to show similar problems (and gave the same error messages). When I started pruning my big program to an excerpt I could post, I used CWEB rather than CWEBx just because it seems more people have had experience with the former. I just tried CWEBx on both the excerpt and on the original program. CWEBx does handle the excerpt properly, but it still doesn't handle the full program.

A complete list of all reserved words appears in section 28 of the Levy/Knuth CWEAVE listing (note that all C++ keywords are there; to use them as identifiers in C requires similar precautions as for |error|). For CWEBx they are in section 117, but you needn't look it up; the only anomaly is |va_dcl| which is there for historic reasons (types that are defined in certain ANSI header files, like |FILE|, are also predefined in CWEAVE, whether or not you include that header file).

Argh. Why isn't this part of the user documentation?

I don't understand this. First of all a ``section in limbo'' is a contradiction in terms, since limbo is the TeX text before the first section.

Apologies for a misuse of a technical term.

I assume you meant a section that is defined (or cited) but never used. This causes a warning message by CWEAVE both in CWEB and CWEBx, since it could indicate an oversight or typing error on the part of the programmer, but it should not otherwise affect the output of CWEAVE. And CTANGLE doesn't care about unreferenced modules at all, although it will complain about undefined ones. Certainly it should make no difference whether a module is defined before it is used or the other way around: both are perfectly valid (although the former is a bit less customary), and should lead to well-formatted output.

Sounds good, if it is just a matter of warning messages. I came by my impression after seeing error messages such as:

```
This is CWEAVE (Version x2+1.2a)
*1
! Never used: <should we be trying to read?>
Writing the output file...*1
! You need an = sign after the module name. (l. 16)
@<should we be trying to read?@>;
```

Since my last post I discovered that this was due to my using @p to introduce the main program rather than @c. (The documentation says they are equivalent; experience says otherwise. :-( ) Thanks again for your help. I will continue trying to put my program into a form that produces the kind of output I want and documenting my problems along the way. Perhaps, in the end, I'll have enough confidence in the tools to try to get others to use them.

Making noweb produce ASCII text

From: **Joseph Brothers**
Date: 10 Dec 1994

Can anyone suggest a way to derive formatted ASCII text from noweb source? The answer may be as simple as locating a TeX-to-nroff filter. Nroff-style output would be fine. I want to use noweb to write a set of project documents jointly with 9 other individuals dispersed across two continents. We have only email in common (flat ASCII). We need noweb's outlining, chunking, indexing and cross-referencing capabilities but we can't exchange noweb source, LaTeX, dvi, or PostScript. We can only exchange

ASCII text.

We can't exchange noweb source because my collaborators do not read or write noweb and many of their computers will not support noweb, HTML, LaTeX, or TeX without more time and expense than the project will bear. PostScript and dvi are very large and too difficult to mark up. I hope to demonstrate literate programming is useful in distributed collaborative authorship projects with a very low level of common software support. I will post a solution here if one is found.

From: **Lee Wittenberg**
Date: 12 Dec 1994

Joseph Brothers writes: Can anyone suggest a way to derive formatted ASCII text from noweb source? The answer may be as simple as locating a TeX-to-nroff filter. Nroff-style output would be fine.

As I understand it, Norman Ramsey has a standing offer to build a troff/nroff back-end for noweb if someone will provide specs. You might want to do that (or whip up a back-end of your own).

From: **Peter Knaggs**
Date: 16 Dec 1994

Joseph Brothers writes: Can anyone suggest a way to derive formatted ASCII text from noweb source? The answer may be as simple as locating a TeX-to-nroff filter. Nroff-style output would be fine. I want to use noweb to write a set of project documents jointly with 9 other individuals dispersed across two continents. We have only email in common (flat ASCII). We need noweb's outlining, chunking, indexing and cross-referencing capabilities but we can't exchange noweb source, LaTeX, dvi, or PostScript. We can only exchange ASCII text.

Get noweb to produce LaTeX output, but include the dvidoc style option. Then when you are ready to email your software process it though noweb, and LaTeX. This should now leave you with a .dvi file. Pass this through the dvidoc program and you have a nicley formatter plain text file. You can then send the .txt file as your software. Dvidoc is available from your local CTAN site.

From: **Fariborz Tavakkolian**
Date: 16 Dec 1994

Joseph Brothers writes: Can anyone suggest a way to derive formatted ASCII text from noweb source? The answer may be as simple as locating a TeX-to-nroff filter. Nroff-style output would be fine.

Although not directly related to noweb, I thought it might be worth a mention. In "Literate Programming" (D. E. Knuth, ISBN 0-937073-80-6 (paper)), page 133, paragraph 1 "As a result of this experience I think it's reasonable to state that a WEB-like system can be create from scratch in a fairly short time, for some other pair of languages besides TeX and Pascal, by an expert system programmer who is conversant with both languages."

As is described earlier in the paragraph, "a short time" is approximately eight weeks. Later in the same paragraph a reference is made to "CWEB " (not CWEB) by Harold Thimbleby (University of York, August 1983) which is based on Troff/Nroff and C. A quick search via anonymous ftp at the author's last known address (ftp.york.ac.uk) revealed nothing. Another option may be CWEB or Spidery WEB modified to use Texinfo (GNU distribution). To this untrained eye, it seems possible.

From: **Colman Reilly**
Date: 16 Dec 1994

Joseph Brothers writes: Can anyone suggest a way to derive formatted ASCII text from noweb source? The answer may be as simple as locating a TeX-to-nroff filter. Nroff-style output would be fine. I hope to demonstrate literate programming is useful in distributed collaborative authorship projects with a very low level of common software support.

Hmmm. I think that if you go latex->noweb->html->text you might get away with it. You can't use any of latex's fancy graphics anyway if you're going to be exchanging ASCII. The last step (html->text) could be done using either lynx or the line-mode browser from CERN. I'd recommend using latex2html with the file splitting turned off!

From: **Joseph Brothers**
Date: 09 Jan 1995

This little answer is produced using itself, that is, what you are reading in the newsgroup or mailer was authored in noweb and postprocessed into formatted text. The noweb source is included as a uuencoded gzip file. If you have noweb, you might choose to create .ps format output. You can read either of these formats, follow its instructions, and reproduce the original posting.

1. Yet Another Frequently Asked Question (FAQ). This is the question as originally posted to comp.programming.literate: Can anyone suggest a way to derive formatted ASCII text from noweb source? The answer may be as simple as locating a TeX-to-nroff filter. Nroff-style output would be fine.

2. Some Frequently Offered Suggestions (FOS). I received a number of interesting and informative replies while I was prowling about with ftp. They amounted to:

1) Norman Ramsey [noweb author] has a standing offer to build a troff/nroff back-end for noweb if someone will specify and use it.

2) Use dvidoc from CTAN.

3) Use dvi2tty from CTAN; in the US it's pip.shsu.edu (192.92.115.10).

4) Use dvi2tty from sol.cs.bucknell.edu:droms/txt-dist.tar.

2.1 Don't Ask Norman To Build One. I don't know enough about either roff to specify a new backend for noweb, and Norman has commented before how nasty cross-referencing would be in nroff. Also, this could take a while.

2.2 Don't Use dvidoc. dvidoc from CTAN is a Pascal program. I don't have Pascal. I don't want it. I've reformed. Also, this is the oldest translator from .dvi to .txt I could find.

2.3 Don't Use dvi2tty from CTAN. dvi2tty from CTAN is an improved C version of dvidoc. I tried it. It produces pages that are a little ragged. On the other hand, you do get your pagination, including header text and page numbers, and a fairly decent looking table of contents. On the other, other hand, code chunk labels get almost too ugly to recognize because the ``<<", ``>>", and ``=" are converted to something else. Shell script and C lose readability too, because ``" and ``" get lost, as does ``:". As a formatter of noweb literate C programs, this is not going to work.

2.4 Use dvi2tty from sol.cs.bucknell.edu To get the tool that works, ftp to sol.cs.bucknell.edu (134.82.1.8). The version of dvi2tty that works best with noweb is in /droms/txt-dist.tar. It's pretty simple to compile, just check the README. To use it, add the two included style guides to the \\documentstyle after all others. Make sure txt is the very last style listed. It sets most everything back to 12 point type.

```
% for typesetting
%\documentstyle[noweb,twoside]{article}
% for ASCII text
\documentstyle[noweb,twoside,myparms,txt]{article}
```

With the text style guides in place, use noweb as usual, allowing it to default to LaTeX output. This will produce a .dvi file for dvi2tty. Use dvi2tty with the undocumented -e option to alter the width of spaces. These are the commands that produced the text you are reading.

```
noweb notext.nw
latex notext
latex notext
dvi2tty -e12 notext -onotext.txt
```

With a negative value the number of spaces between words becomes less. With a positive value it becomes more. It may be needed to reduce the prevalence of words jamming together. The problem is worse without -e. -e12 worked well enough for me. Margins in the text output seem a bit uncertain, too.

3. Future References. This may all be moot. Two other toolsets capable of both typeset and text output from the same source exist. Neither is yet integrated with a literate programming tool, but that may change. Here is a scrap from ftp.cs.cornell.edu:/pub/mdw/Linuxdoc-SGML.README. "Linuxdoc-SGML is a text-formatting package based on SGML (Standard Generalized Markup Language), which allows you to produce LaTeX, groff, HTML, and plain ASCII (via groff) from a single source. Texinfo support is forthcoming; due to the flexible nature of SGML many other target formats are possible. It's easy to learn and use and greatly simplifies writing documents to be produced in multiple formats."

How to distinguish x from x.* in indices (noweb)

From: **Balasubramanian Narasimhan**
Date: 19 Dec 1994

I can't seem to find any info on how to make noweb distinguish an index entry

```
@ %def params

from

@ %def params-print-format
```

I end up getting indices that say params is used in a chunk when only params-print-format is actually used (although, this still kind of makes sense). Help anyone? Thanks.

From: **Jacob Nielsen**
Date: 20 Dec 1994

Balasubramanian Narasimhan writes: I can't seem to find any info on how to make noweb distinguish an index entry @ %def params from @ %def params-print-format I end up getting indices that say params is used in a chunk when only params-print-format is actually used (although, this still kind of makes sense).

As other language independent tools noweb doesn't know that `params' and `params-print-format' are two different things and it is quite difficult/impossible to come up with a method of determining what is what without knowing the language. (e.g. in C `params-print-format' would be an equation !) The use of whitespaces to seperate tokens is not sufficient, and for most languages characters such as -, +, = etc. does seperate tokens.

However, noweb has the ability to know something about the language with the use of filters (see the noweb hackers guide). If you're programming in lisp and want automatic insertion of `@ %uses params-print-format' instead of what you get now: `@ %uses params' where `params-print-format' is used, you have two options: 1. Write your own filter. 2. Drop the automatic detection of uses and insert them yourself.

From: **Lee Wittenberg**
Date: 20 Dec 1994

Balasubramanian Narasimhan writes: I can't seem to find any info on how to make noweb distinguish an index entry @ %def params from @ %def params-print-format I end up getting indices that say params is used in a chunk when only params-print-format is actually used (although, this still kind of makes sense).

The problem occurs because noweb considers `-' to be a symbol that can end an identifier rather than a necessary part of the identifier, so the recognizer heuristic gets confused. The simple solution is to use an underscore in your identifiers rather than dashes (if you can).

A slightly trickier solution is to change the definitions of ALPHANUM and SYMBOLS in finduses.nw so that the former includes `-' and the latter doesn't. Then retangle and recompile finduses, and you've got a noweb that recognizes identifiers the way you want. This may cause some problems if you don't put spaces around a minus sign in expressions, but languages (like COBOL) that allow dashes in identifiers usually require spaces around operators.

A tougher, but more robust, solution is to write a filter (or a pair of filters) that adjusts things to use underscores for indexing and then change them back before the back end can typeset the index entries.

From: **Preston Briggs**
Date: 20 Dec 1994

Balasubramanian Narasimhan writes: I can't seem to find any info on how to make noweb distinguish an index entry @ %def params from @ %def params-print-format I end up getting indices that say params is used in a chunk when only params-print-format is actually used (although, this still kind of makes sense).

Noweb (and nuweb) believe that "-" is an operator rather than part of an identifier. Thus, noweb thinks "params-print-format" is 3 identifiers. An embarrassing language sensitivity in an otherwise fairly language-indipendent tool. You might rewrite as "params_print_format" or start reading the code. There's a place (I know not where) that specifies a string of characters that may belong to an identifier. Just edit and recompile. Of course, then you'll start seeing things like "x-y" treated as an indentifier when you intended then to be treated separately. Sigh.

From: **Norman Ramsey**
Date: 20 Dec 1994

Balasubramanian Narasimhan writes: I can't seem to find any info on how to make noweb distinguish an index entry @ %def params from @ %def params-print-format I end up getting indices that say params is used in a chunk when only params-print-format is actually used (although, this still kind of makes sense).

Indeed. For example, in C, Standard ML, and many other languages this would be correct behavior. Noweb uses a simple, language-independent heuristic to determine what an identifier is. Characters are classified as Alphanumerics, Symbols, and Delimiters. An identifier is (approximately) a string of alphanumerics, delimited by symbols and delimiters, or a string of symbols, delimited by alphanumerics and delimiters. The heuristic, when combined with appropriate choices of alphanumerics, symbols, and delimiters, does a good job for a wide range of languages. The noweb filter 'finduses' implements the heuristic. Your options are:

1. Write a language-dependent version of finduses, which can implement your languages precise rules for identifiers (e.g., you can avoid false hits in string literals and in comments),

2. Arrange for finduses to believe that - is an alphanumeric. Or if your language is Scheme, you could make everything except parens, white spacem and quotes alphanumeric. Or whatever. An elegant way to do this would be to make it possible to specify these sets on the command line using typical Unix character-set notation.

From: **Balasubramanian Narasimhan**
Date: 23 Dec 1994

I want to thank all the folks for the suggestions. I finally ended up hacking [[finduses.nw]] so that it reads a ./.nowebrc file if one exists for ALPHANUMS and SYMBOLS. Otherwise, it uses the default. Took about 10 minutes and works beautifully.

Compliments to the developers of noweb and nuweb!

From: **Ben-Kiki Oren**
Date: 01 Jan 1995

I am looking for advice on how to begin integrating literate programming tools into my development environment. My situation is as follows: I work in a small start-up, where in the last year or so I was the sole code-writer. I wrote approximately 20K LOC (in C++), 15K of which are in a major library (~150 .cpp files). The code is reasonably commented, but lacks documentation of the overall structure and the relationships between the different library parts. We are now expanding and hiring a few (~4) more programmers. I therefore need to complete the documentation. Further, in my new role as 'lord of code writers', I need to set up coding guidelines (and my code had better satisfy them :-)

Now, my code is rather orderly, and follows a consistent style (which I can formalize without much trouble). However, as I mentioned above, its documentation is somewhat lacking. I have been following this newsgroup about a year ago, but I never used a literate programming tool. I thought this would be a good opportunity to start using one; if it works out, I could have have all my programmers use it, and make it an integrated part of the development process.

The problem is, of course, which tool to use. We will be developing our libraries on UNIX. The GUI will be based on MS/Windows, but I feel that complex documentation of the GUI part is unnecessary, since the help system will fill the same need. So, I can settle for a UNIX-based tool.

I would naturally like the tool I use to be as simple and lightweight as possible. I would also like to have hyper-text abilities, good indices, etc. Formatting should be kept simple, so I am weary of tools which require using plain TeX. LaTeX is probably acceptable; Texinfo would be perfect, if it weren't for the strange fact the FAQ did not list any tool using it. Some may be configured for it, I suppose, but I lack the time for prolonged customizations.

The trouble is, quite a few tools mentioned in the FAQ may be appropriate, in particular:

- CLiP (may be configurable to using Texinfo, but did it say a 64 files limit?)
- CWEB (integrated C/C++ support)
- FunnelWeb (to quote the FAQ: 'production-quality' + 'simple', plus tutorials)
- FWEB (how come its documented in Texinfo, but produces TeX/LaTeX?)
- noweb (very simple, HTML support, could be just the thing)
- nuweb (simple)
- ProTeX? (well, someone published a book on it, so it can't be that bad :-)

Obviously I can't just download each, play with it for a week, and make up my mind. I will probably download two, or at most three, and make up my mind after reading the docs. So, could anyone give me some good arguments for/against any of the above, point out

the major differences, or otherwise help me out? I'll post a summary of the replies; if its relevant, I'll format it as an addition to the FAQ.

From: **Joseph Brothers**
Date: 01 Jan 1995

Please try noweb. It's not only very simple in operation and easy to learn to use, it is very powerful and has many useful output options including a very nice HTML converter. Its language independence and LaTeX integration make it broadly useful. Noweb quickly becomes familiar and gets out of your way, letting you improve the clarity of your understanding and exposition of your code.

From: **Ben-Kiki Oren**
Date: 16 Jan 1995

First, I would like to thank all the people who answered my query. I have received quite a few answers (literate programming must be popular :-).

I found that the trick to extracting useful information from ads is to look for the description of the missing features in the competitor's products. (I assume that if a company dares to advertise a deficiency in another product, they are willing to back it up in court). By trivial extension, I payed more attention the problems described for each tools then to the strengths. To prevent the flame-wars that would result, I'll omit the more blunt descriptions. In essence, most tools were described as too complicated / hard to learn / hard to use and / or technically deficient in some major way.

Surprisingly, there were two tools whose only faults mentioned was the lack of pretty printing and automatic indexing for C++. Pretty-printing I can live without. I suppose manual indices can be annoying, but it is minor compared to some of the faults ascribed to the other tools. My compliments to the developers of noweb and nuweb!

I'm loading them up now, and will look into both in the next few days. I am a bit hazy on how to integrate them into my environment. Literate programming in C++, using templates and multiple inheritance, seems to call for great care in the partitioning of the source into files (one per class? two per class? how to resolve references to base classes and templates? how to prevent unnecessary 'touch'es of the header files? etc.) There were a few hints on the literate programming html WEB page; I'd appreciate any further advice or references on the subject.

CWEB: Output without c-code

From: **Claudia Hattensperger**
Date: 11 Jan 1995

Who can help me? To get a good documented C-code, I'd like to use a literate-programming tool to have documentation and C-code together in one file and I tried CWEB together with the LaTeX-style (cweb.sty). All worked very well, apart I didn't succeed in getting only the documentation without the C-code as output from cweave. As the program is part of a master thesis, and it is unusual to include all the code in a thesis, I'd like to have the possibility telling cweave to ignore special procedures or all code.

I have tried: 1. Defining \def\ignore#1{} in the limbo and the text following '@c' as argument. But I couldn't find a place for the closing bracket: before the next '@ ' was impossible because the compiler couldn't parse the code produced by ctangle, after the next '@ ' didn't work as ctangle complained. 2. Changing cweb.sty or cwebmac.tex. Who can tell me a solution to my problem?

From: **Przemek Klosowski**
Date: 13 Jan 1995

Claudia Hattensperger writes: To get a good documented C-code, I'd like to use a literate-programming tool. As the program is part of a master thesis, and it is unusual to include all the code in a thesis, I'd like to have the possibility telling cweave to ignore special procedures or all code.

And what is wrong with including source code in your thesis? My PhD thesis had code in it, allright, and it wasn't even about computer science (physics, actually). I structured it such that the code ended up in an appendix.

From: **Norman Ramsey**
Date: 19 Jan 1995

Claudia Hattensperger writes: Who can help me?

I can :-)

As the program is part of a master thesis, and it is unusual to include all the code in a thesis, I'd like to have the possibility telling cweave to ignore special procedures or all code.

I've set up a noweb filter to do this in noweb. It omits any code chunk whose name matches a globbing pattern given on the command line. I have found to to be useful for writing papers in which it is not appropriate to show all the code. This `elide' filter will appear in the 2.6d distribution, which I'll release the moment I believe I have half an hour free to ship it. Be warned: noweb does no prettyprinting; if you want soething other than \tt font, you need Kostas Oikonomou's prettyprinting package (a new version of which is delaying 2.6d...), which is distributed with noweb.

From: **Frank Niessink**
Date: 01 Mar 1995

Let's say I want to program a C++ library. I want to use a literate programming tool to do this. I would prefer to use noweb, cause I already know it a little, but if any other tool offers what I need I'll happily switch. My questions are: 1) is it possible to hide code from the documentation? 2) and is it possible to duplicate code? For example, imagine I want to describe the following class interface:

```
class some_class
{
private:
  some_type some_var;
public:
  some_class();            // constructor
  ~some_class();           // desctructor
  some_member();
}
```

But in my documentation I want to have the following:

```
class some_class
{
public:
  some_class();            // constructor
  ~some_class();           // desctructor
  some_member();
}
```

And in an appendix I want the complete class definition. So part if it will be in the documentation twice. Is that possible, and if so, how?

From: **Norman Ramsey**
Date: 04 Mar 1995

Frank Niessink writes: My questions are: 1) is it possible to hide code from the documentation? 2) and is it possible to duplicate code?

I'm not sure I inderstand either question. I sometimes hide whole code chunks using noweb's elide filter. E.g.

```
<<*>>=
class some_class
{
  <<mysterious aspects>>
public:
  some_class();            // constructor
  ~some_class();           // desctructor
  some_member();
}
@ where the definition of <<mysterious aspects>> is hidden from weave
but not tangle.
```

But in my documentation I want to have the following:

```
class some_class
{
public:
  some_class();            // constructor
  ~some_class();           // desctructor
  some_member();
}
```

You could easily write another filter to cause the line containing the use to vanish as well.

And in an appendix I want the complete class definition. So part if it will be in the documentation twice.

I would handle this by replicating the documentation, i.e., use the same noweb source to build two different .tex files, one to be used in the body of the document, and one to be used in the appendix.

From: **Dave Love**
Date: 06 Mar 1995

Frank Niessink writes: 1) is it possible to hide code from the documentation?

Yes, formatter-dependently. For LaTeX, there are comment environments available, e.g. in the `verbatim' package, which provide a general solution and similar things are obviously possible in plain TeX. For a formatter which didn't support multi-line comment blocks you could presumably insert another filter in the pipeline to do the job.

From: **Tony Coates**
Date: 06 Mar 1995

Does noweb support unused code-blocks? In FunnelWeb, which I use, a block of code can be marked as unused, so that it is typeset like a code section, but is not used in the generation of any files. You would then put the full class definition in the appendix, and a *phony* definition earlier in the documentation. This has the problem that you have to manually synchronise the two definitions. An alternative with FunnelWeb would be to define the public definitions as a macro earlier int he text, and invoke that macro in the true definition int the appendix. The full code wouldn't appear in the appendix then, a reference to the original macro appearing in its place. Is this anything like what you want to do?

From: **Frank Niessink**
Date: 07 Mar 1995

*Tony Coates writes: Does noweb support unused code-blocks? In FunnelWeb, which I use, a block of code can be marked as unused, so that it is typeset like a code section, but is not used in the generation of any files. You would then put the full class definition in the appendix, and a *phony* definition earlier in the documentation.*

Yes. noweb does support hiding chunks, as Norman Ramsey pointed out to me. One can use the filter elide to hide specific chunks from the output. It should also be prossible to create two documents from one source, one with the body and one containing the appendix (with the complete source code). But I haven't figured out yet how to do that. Norman Ramsey suggested I should use the same source here to create two different .tex files, one with the body and one with the appendix. It seems to me that that would be quite a complicated task. Could anyone shed some light on that?

This has the problem that you have to manually synchronise the two definitions. An alternative with FunnelWeb would be to define the public definitions as a macro earlier int he text, and invoke that macro in the true definition int the appendix. The full code wouldn't appear in the appendix then, a reference to the original macro appearing in its place. Is this anything like what you want to do?

It is, maybe I should check out FunnelWeb too.

From: **Eitan Gurari**
Date: 07 Mar 1995

Frank Niessink writes: Let's say I want to program a C++ library. I want to use a literate programming tool to do this. I would prefer to use noweb, cause I already know it a little, but if any other tool offers what I need I'll happily switch. My questions are: 1) is it possible

to hide code from the documentation? 2) and is it possible to duplicate code? For example, imagine I want to describe the following class interface:

```
class some_class
{
private:
  some_type some_var;
public:
  some_class();           // constructor
  ~some_class();          // desctructor
  some_member();
}
```

But in my documentation I want to have the following:

```
class some_class
{
public:
  some_class();           // constructor
  ~some_class();          // desctructor
  some_member();
}
```

And in an appendix I want the complete class definition. So part if it will be in the documentation twice. Is that possible, and if so, how?

The following example illustrates how ProTeX can be used to achieve the above objectives.

```
%---------------------- example --------------------------------------

\input ProTex.sty
\AlProTex{cpp,<<<>>>,list,title,`}

\long\def\condshow#1\?{}
\def\ShowChunk\<#1\>{{\let\condshow=\empty
   \csname :DoName\endcsname\ShowCode{#1}}}
\let\svModifyOutputCode=\ModifyOutputCode
\def\ModifyOutputCode{\let\?=\empty \svModifyOutputCode}
\let\svModifyShowCode=\ModifyShowCode
\def\ModifyShowCode{\let\?=\condshow \svModifyShowCode}

Main
====

\<chunk\><<<
class some_class
{`?
private:
  some_type some_var;`?
public:
  some_class();           // constructor
  ~some_class();          // desctructor
  some_member();
}
>>>

Appendix

\ShowChunk\<chunk\>
\OutputCode\<chunk\>
\bye
```

From: **Mark Kramer**
Date: 08 Mar 1995

Frank Niessink writes: Let's say I want to program a C++ library. I want to use a literate programming tool to do this. I would prefer to use noweb, cause I already know it a little, but if any other tool offers what I need I'll happily switch. My questions are: 1) is it possible

to hide code from the documentation? 2) and is it possible to duplicate code? For example, [...]

The first point is very easily solved with CLiP. Just use a separate source file (=CliP-input) for those parts you don't want to see in the main documentation. To extract the complete program, CLiP these source files together. We use this method frequently in CLiP itself. Maybe, this is a solution for the second point as well? We have experience with such an approach in Modula-2, where the specification files and the implementation files were generated from the same set of source files. But I realize that this is no answer to the second point as it stands: the interface definition occurs only once in the documentation.

However, that is how it should be in a literate program. One should not want to document the same thing at two separate places. In my opinion, the very idea of literate programming is to keep together what belongs together. Nevertheless, I have sometimes experimented with this idea of 'double documentation'. If your wordprocessor (or formatter) supports include-files, this is easy too.

From: **Frank Niessink**
Date: 08 Mar 1995

Mark Kramer writes: But I realize that this is no answer to the second point as it stands: the interface definition occurs only once in the documentation. However, that is how it should be in a literate program. In my opinion, the very idea of literate programming is to keep together what belongs together.

OK, so blame it on C++. But in the documentation of a C++ library I definitely want to document the same thing on two places. One could even argue I want to document a class on three different levels: 1) the class interface for users of the class 2) the class interface for those who might want to inherit from the class (i.e. more attention for the way the class works, internally). 3) the complete class interface.

I want to use literate programming as a tool that enables me to write code and documentation that is tightly coupled and helps me to keep documentation and code consistent and up-to-date. I don't want it to restrict the way I work. I strongly disagree with you, if you say: "One should not want to document the same thing at two separate places." for the reasons I mentioned above. Please let me hear what you think. I think this is an interesting subject and I'd like to hear your opinions.

From: **Mark Kramer**
Date: 08 Mar 1995

Frank Niessink writes: OK, so blame it on C++. But in the documentation of a C++ library I definitely want to document the same thing on two places. One could even argue I want to document a class on three different levels: [...]

The two/three *levels* is not the problem, the two/three *places* are. Or am I missing something?

I want to use literate programming as a tool that enables me to write code and documentation that is tightly coupled and helps me to keep documentation and code consistent and up-to-date.

IHMO, this will only work if you keep the different levels close together. See also my posting on "meta literate programming".

I don't want it to restrict the way I work.

Does that also mean: don't want to change ...?

From: **Frank Niessink**
Date: 09 Mar 1995

Frank Niessink writes: OK, so blame it on C++. But in the documentation of a C++ library I definitely want to document the same thing on two places. One could even argue I want to document a class on three different levels: [...]

*Mark Kramer writes: The two/three *levels* is not the problem, the two/three *places* are.*

No. You are entirely right here. I shouldn't have used the word "levels". The problem lies in the different places where I want to say different things about (partly) the same code. See also below.

Frank Niessink writes: I don't want it to restrict the way I work.

Mark Kramer writes: Does that also mean: don't want to change ...?

Give me some credit, please. If I would start to use literate programming, that on its own would be a change in the way I work. Maybe I have conflicting needs or I have some wrong ideas of what literate programming is, or maybe both. I don't know, if you do, please tell me. I still think it makes good sense to write the documentation as I described before and I want to know whether literate programming will help me with that or not. If it is not possible, well, just say so. If it is not in the spirit of literate programming, explain to me why. But don't imply that I am not willing to change and that it is my fault if literate programming doesn't offer what I want. Maybe literate programming just isn't suitable for libraries?

Sorry if I sound a bit irritated, but I'd like to see people play the ball, not the player.

From: **Frank Niessink**
Date: 09 Mar 1995

Frank Niessink writes: Yes. noweb does support hiding chunks, as Norman Ramsey pointed out to me. One can use the filter elide to hide specific chunks from the output. It should also be prossible to create two documents from one source, one with the body and one containing the appendix (with the complete source code). But I haven't figured out yet how to do that. Norman Ramsey suggested I should use the same source here to create two different .tex files, one with the body and one with the appendix. It seems to me that that would be quite a complicated task. Could anyone shed some light on that?

All much easier than it sounds. In your main document, you may have

```
...
\include{body}
\appendix
\include{appendix}
...
```

Then build body and appendix like this:

```
noweave -filter "elide 'hidden:*'" source.nw > body.tex
noweave                            source.nw > appendix.tex
```

If you don't want the word 'hidden' to appear in the names of the appendix chunks, you could use

```
noweave -filter "elide 'hidden:*'"               source.nw > body.tex
noweave -filter "sed 's/^@defn hidden:/@defn /'" source.nw > appendix.tex
```

Perhaps one day someone will write a noweb tutorial explaining these and many other tricks :-)

From: **Jens Horstmeier**
Date: 21 Aug 1995

I've been working with both cweave and noweave and I like these tools quite a lot. There is one point I really miss with them, the abilty to create interface specifications out of the input file. Suppose I am writing a module for a larger project and I am working together with several other programmers. It would be nice to give them a document, that contains all the functions, classes, ... I am exporting to them without the need to give them the whole implementation doc. I don't want to have multiple source documents, since this does not really work and the specification must already be written in the implementation doc anyway. So my question: Is there a tool to extract specific parts of a literate program to build up an interface specification.

From: **Norman Ramsey**
Date: 23 Aug 1995

Jens Horstmeier writes: Is there a tool to extract specific parts of a literate program to build up an interface specification.

I've done tolerably well with a combination of TeX conditional formatting and heavy use of the elide filter that comes with noweb. I've been only mildly satisfied with the results...

From: **Come Raczy**
Date: 24 Aug 1995

Jens Horstmeier writes: Is there a tool to extract specific parts of a literate program to build up an interface specification.

With FWEB, it's possible to construct easily your document s.t. it's easy to extract only a part of it (there is a powerfull preprocessor which may help).

Newbie question (not from AOL)

From: **Eric Dahlman**
Date: 24 Jan 1995

I understand the concept of literate programming in a somewhat abstract sense and I think that I was going about it from the wrong direction. First off I am very impressed with the way that documentation and code are mixed in .dtx files in LaTeX and the results that are produced by these files. Now I was wondering if and how one would go about finding a similar system for non-(La)TeX files. I looked at some of the web files on CTAN and to be honest I was a bit confused. Where does one start? What bits are necessary and which are just extra fluff? So would one of the kind soles that reads this group help a babe-in-the-woods and tell me how to get the whole thing up and running.

From: **Weiqi Gao**
Date: 24 Jan 1995

The way I started on literate programming is through the original WEB system by Knuth. It is included in the emTeX package for DOS. The document in the package is an excerpt from a technical report, so it is terse. One drawback at the time is that I don't have a PASCAL compiler, so I just wrote some junk webs and ran them through WEAVE and TANGLE. WEAVE outputs a *.tex file which, when processed and printed out, is a beautifully formatted document. TANGLE produced a PASCAL program that looks more like a uuencoded executable.

If you want to get the original ideas of the great one, get the original WEB for PASCAL and just try to figure it out. I guess no-body writes in PASCAL any more, so there won't be any practical use of it. However if you try to learn any other literate programming tool, the first paragraph of the documentation will invariably have a reference to the original WEB, saying something like: "The original WEB is great, but it is for PASCAL only..." or "The original WEB is great, but it has `TOO MANY NOTES'...". So you will be at a loss if you don't know what the original WEB is. You can see the output of the original WEB everyday: the TeX/Metafont system was written in WEB---the programs you are using, and the `TeX: the program' you are reading are produced by WEB.

Then you go ahead and choose a real literate programming tool. And here is where the fun begins. If you are like most other people---who discovered literate programming by themself and not by a department policy---you will go around and evaluate a list of tools available on the net, and arrive at the one that works. The one for me is noweb. (Don't ask why, or you'll start religious war #19.) For other people it might be different. I read an interview with DEK on WWW (O'reily pages?), in which he claims CWEB is the best system if you're programming in C.

The biggest trouble for literate programming is that the tools aren't used on a day to day basis. There is also a tendency for everybody to write his own literate programming tool. Another hassle for using a literate programming tool is that they are usualy not ell integrated with the development environment, especially the debugging tools.

Once you've decided which package to try---be it CWEB, FWEB, noweb, nuweb, FunnelWeb, Spider Web---you can usually get the source from CTAN. Installation is usually easy, unless you have to use DOS. Executables for DOS are usually available. Each literate programming tool is usually written in its own brand of WEB. The LaTeX3 projects doc system differs from an ordinary literate programming system in that with it, the text-processor that produces the documentation and the `compiler' that produces the `executable' are the same system.

Here again, in summary, are the steps people take on literate programming:

1. Be impressed by a LitProg tool---in particular the TeX-ed out documentation.
2. Experiement with different tools.
3. Settle down on noweb, and don't think about it any more.

From: **Norman Ramsey**
Date: 24 Jan 1995

Well, let me just dust off my flame-proof suit... (David Thompson, I think some of these recommendations ought to go in the FAQ.)

Some of us think that .dtx and docstrip omit an essential capability, viz., that of writing fragments of code in any order. Religious war #21?

Unlike Weiqi Gao, I suggest a good place to begin is in your library, not by playing around with different tools. ``Sometimes two weeks' hacking can save a whole afternoon in the library.'' Don Knuth has written a book entitled ``Literate Programming,'' which I think is mostly of historical interest, but does contain a smorgasbord of good stuff, including the original 1984 Computer Journal article that started this business. Don's ``Stanford GraphBase'' book contains literate programs that I find more accessible than TeX and METAFONT.

But the easiest way to begin is probably to explore back issues of Communications of the ACM, where you can find John Bentley's columns on literate programming, plus the columns moderated by Chris van Wyk. (It's not desirable to read all the van Wyk columns unless you want an idea of the range of crazy things people [including me!] have tried and called literate programming.) If you want to understand the issues a little more deeply, you can read my SPE article, which describes some problems encountered using WEB in practice, and my IEEE Software article, which explains how you can get the features you want without having to use a complicated tool. Spend an afternoon this way and you will be the local expert on literate programming (bibliography at end).

Weiqi Gao writes: The biggest trouble for LitProg is that the tools aren't used on a day to day basis.

Huh? Evidence? Tools range in popularity from daily used worldwide to dead and best forgotten.

There is also a tendency for everybody to write his own literate programming tool.

Yes. It's much more easy and fun to write a literate-programming tool than to write a literate program.

Another hassle for using a literate programming tool is that they are usualy not well integrated with the development environment, especially the debugging tools.

Some are, some aren't. To my mind the major lack of integration is not with compiling and debugging tools, but with WYSIWYG word processors. CLiP comes close, but at the cost of some ugly syntactic conventions (in the eye of this beholder), and without the ability to support automatic indexing.

Once you've decided which package to try---be it CWEB, FWEB, noweb, nuweb, FunnelWeb, Spider Web

For the beginner, there are only three choices. Sensible people will choose noweb or nuweb. Those who are seduced by Don's pretty C code will choose CWEB. (My article in Software explains why.)

Selected bibliography: @string{CACM="Communications of the ACM"}

@article{bentley:lp1, month=may, pages="364--368", number="5", author="Donald E. Knuth and Jon L. Bentley", title="Programming Pearls: Literate Programming", keywords="literate programming pearls", annote="Bentley introduces literate programming. Knuth gives solution to: print a sort list of M random numbers in the range $1\ldots N$, with no duplicates. Solution in exercise 3.4.2--15 in {\it Seminumerical Algorithms}. Uses ordered hash tables.", journal=CACM, year="1986", volume="29"}

@article{bentley:lp2, month=jun, pages="471--483", number="6", author="Donald E. Knuth", title="Programming Pearls: A Literate Program", journal=CACM, year="1986", note="Reviewed by M. Douglas McIlroy", annote="Knuth's ornate solution to the common words problem. Introduces the hash trie data strucuture. Review by McIlroy.", volume="29"}

@article{knuth:literate, pages="97--111", number="2", author="Donald E. Knuth", title="Literate Programming", annote="The original article on web and literate programming.", journal="The Computer Journal", year="1984", volume="27"}

@article{gries:pearls, month="April", pages="284--290", number="4", author="David Gries and Jon Bentley", title="Programming Pearls: Abstract Data Types", keywords="web literate programming", journal=CACM, year="1987", volume="30"}

@article{cvw:assessment, month="March", pages="361--365", number="3", author="Van Wyk, Christopher J.", title="Literate Programming: An Assessment", journal=CACM, year="1990", volume="33"}

@article{thimbleby:review, month=jun, pages="752--755", number="6", author="Harold Thimbleby", title="A Review of {Donald} {C}. {L}indsay's Text File Difference Utility, {{\em diff}}", journal=CACM, year="1989", volume="32"}

@article{ramsey:literate, author="Norman Ramsey and Carla Marceau", title="Literate Programming on a Team Project", keywords="web", journal="Software---Practice \& Experience", month=jul, volume=21, number=7, pages="677--683", year="1991", refereed=1, also="Princeton tech report CS-TR-302-91"}

@article{ramsey:simplified, refereed=1, author="Norman Ramsey", title="Literate Programming Simplified", journal="IEEE Software", month=sep, pages="97--105", volume="11", number="5", year="1994"}

Books on literate programming or that include literate programs

@book{knuth:literate:book, address="Stanford, California", author="Donald E. Knuth", title="Literate Programming", publisher="Stanford University", annote="A mildly interesting collection of Don's work. Begins with his Turing lecture entitled ``Computer Programming as an Art.'' Has two chapters on structured programming: the ``structured programming with goto statements'' paper, and a structured program to generate all topological sorting arrangements. Follows with the original literate programming paper from the computer journal, then two programming pearls papers (sampling and common words). ``How to read a WEB,'' which first appeared as a preface to TeX: the program (I believe), and some excerpts from the TeX and METAFONT programs. There is a chapter on Mathematical Writing that is transcribed from discussions in a course Don gave at Stanford. Then, the ``errors of TeX,'' including the error log of TeX. Finally, the book closes with a short example of CWEB (word count).", year="1992"}

@book{fraser:retargetable:book, author="Christopher W. Fraser and David R. Hanson", title="A Retargetable {C} Compiler: Design and Implementation", publisher="Benjamin/Cummings", address="Redwood City, CA", annote="compiler as a literate program", year=1995, note="ISBN 0-8053-1670-1, in press" }

These are worth reading only as curiosities, not as guides to good practice

@article{ramsey:building, author="Norman Ramsey", title="{L}iterate Programming: {\hskip 0pt plus 0.5em}{W}eaving a Language-Independent {{\tt WEB}}", journal=CACM, month=sep, volume=32, number=9, refereed=1, pages="1051--1055", year="1989"}

@article{cvw:loom, month="July", pages="593-599", number="7", author="David R. Hanson", note="Reviewed by John Gilbert", title="Literate Programming: Printing Common Words", journal=CACM, year="1987", volume="30"}

@article{cvw:transactions, month=dec, pages="1000-1010", number="12", author="Michael Jackson", note="Reviewed by David Wall", title="Literate Programming: Processing Transactions", journal=CACM, year="1987", volume="30"}

@article{hamilton:expanding, month=dec, pages="1376--1385", number="12", author="Eric Hamilton", note="Reviewed by Don Colner", title="Literate Programming: Expanding Generalized Regular Expressions", journal=CACM, year="1988", volume="31"}

@article{denning:announcing, month=jul, pages="593", number="7", author="Peter J. Denning", title="Announcing Literate Programming", annote="Short announcement of the formation of the literate programming column, which brief history.", journal=CACM, year="1987", volume="30"}

From: **Jeffrey McArthur**
Date: 24 Jan 1995

Weiqi Gao writes: If you want to get the original ideas of the great one, get the original WEB for PASCAL and just try to figure it out. I guess no-body writes in PASCAL any more, so there won't be any prectical use of it.

I use PASCAL WEB all the time. We now have several large projects written in Pascal WEB.

From: **Joachim Schrod**
Date: 26 Jan 1995

Eric Dahlman writes: I understand the concept of literate programming in a somewhat abstract sense and I think that I was going about it from the wrong direction. First off I am very impressed with the way that documentation and code are mixed in .dtx files in LaTeX and the results that are produced by these files. Now I was wondering if and how one would go about finding a similar system for non-(La)TeX files.

MAKEPROG is a system that supports Literate Documentation [*] for arbitrary files, in a fashion similar to Frank's doc system. You can get it from the LitProg Archive, in "ftp://ftp.th-darmstadt.de/pub/programming/literate-programming/independent/". [*] As the author of MAKEPROG, I would like to emphasize that neither my system nor the doc system support literate programming. I consider the support of refinements (i.e., the ability to explain code abstractions in chunks) and thereby to rearrange code an essential part of tools for this programming method. As an example, for TeX programming it would be really handy to have refinements. Lots of contortions might be saved then. If you want to get the feel of a `real' literate programming tool, I would recommend noweb for a start. It's very lean.

From: **Dave Hanson**
Date: 08 Feb 1995

lcc is a retargetable compiler for ANSI C with code generators for the SPARC, MIPS R3000, and Intel x86. The just-published book

```
A Retargetable C Compiler: Design and Implementation.
Christopher W. Fraser and David R. Hanson.
Benjamin/Cummings, 1995. ISBN 0-8053-1670-1.
```

is a detailed tour of the implementation of lcc. The entire book is a "literate program" -- the source code is intermingled with the descriptive text. The book was produced with the noweb, a low-tech literate programming system. Single copies may be ordered directly from the Benjamin/Cummings Order Dept. at 800-447-2226, and are available in bookstores.

lcc is available for anonymous ftp from ftp.cs.princeton.edu in pub/lcc; the README file there gives acquisition details. The distribution includes the source code for the complete compiler and the code-generator generator. More information about lcc is available on the WWW at URL http://www.cs.princeton.edu/software/lcc.

Separate compilation

From: **Ben-Kiki Oren**
Date: 30 Jan 1995

Well, I've downloaded noweb and nuweb off the net and looked into both. Both are nice, but probably noweb is the tool for me since both are (alas!) inadequate to my needs, as it is designed to be easily extendible. I'll probably look into the new FWEB release since it supports HTML (which is my target formatter). I'm ``somewhat unhappy'' with LaTeX2HTML.

At any rate, my problem is that both tools have zilch support for separate compilation. Now, I typically place each C++ class in its own set of .hh/.cpp files (with a possible .ii for inlines, etc.). I naturally want to write a single '.w' file per class. It seems the current way of supporting this is to include all the .tex files into a single global doc and pass it through LaTeX to get a document. This won't do for me, since I've got over a hundred classes, and if I treat the whole thing as a single document, I'll die of coffee overdose. Even worse, I'm not interested in LaTeX at all. I want HTML documents so I'll be able to browse through the docs with Mosaic (who ever prints these docs, anyway? They are out-of-date the second they are printed). LaTeX2HTML would die horribly trying to eat such a monster document.

So, I could simply treat each class as its own separate document, right? Wrong. Because I do want to have reasonable hypertext cross-links between the classes. For example, when one class derives from another, I would like to be able to get to the base class definition. In fact, I would like to be able to click on any class name in the code chunks, and jump to that class definition file. And this currently can't be done when each is in its own document. BTW, I don't have TeX or LaTeX installed at all, and I don't want to. This makes relying on LaTeX-generated .toc files and .aux files impossible.

I've given the matter some thought, and what seems to me the best solution is that each document would generate a list of identifier definitions (with the HTML URL anchors) in a separate '.def' file. In each document, I'll need to say something like '@use some-other-document', which will add these anchors to the identifier definitions table. Thus, when the document is weaved, proper cross-links will be generated. A makefile will make sure the '@use'ed '.def' files are up-to-date.

A secondary issue is the creation of a global table of contents file, which will reference all the separate documents. This can be trivially performed by a separate program picking at the generated HTML files, and creating a list of all the <Hn>...</Hn> entries. I could also generate an automatic use-graph, so I could jump to all places a class was used or derived from, etc... You get the idea.

Simple, in theory. In practice... Well, I'm hampered by the whole LaTeX2HTML mess (I'm desperate enough that writing straight HTML looks really inviting at times). I've just seen the FWEB release post - maybe its use of private formatting commands and HTML support will relieve me of this - but it remains to be seen how easy it would be to integrate such a scheme in it. OK, I'll do it myself if I have to. And if I have the week or two it'll take. I promise to post the result, whichever tool it turns out to be based on.

What really worries me is that the matter was not even briefly mentioned in any literate programming document I've encountered. Is it because no-one uses literate programming for big projects? Come to think of it, the biggest literate program I know of is TeX, and Knuth is just, eh, out-of-the-ordinary enough to write it as a single .w file, for all I know. This is not an option for a team of five programmers working on a library with >30K of C++ LOC and >100 of C++ classes.

So, is there some blessed soul who has already implemented such a scheme? If not, has literate programming ever been implemented for projects with a multitude of cross-referencing source files worked on by multiple programmers? How did they do it, then? Am I missing something obvious?

From: **Norman Ramsey**
Date: 31 Jan 1995

Ben-Kiki Oren has uncovered a flaw in noweb's HTML support (I have been waiting for this shoe to drop for six months), but he has

confused the issue with separate compilation. The problem is that NO current literate-programming tool has good support for inter-document references. (Probably because no TeX variant does this.) Some noweb users have encountered a related but not identical problem: in documentation, chunk names are assumed to denote different `refinements' in different documents.

Ben-Kiki Oren writes: I naturally want to write a single '.w' file per class. I want HTML documents so I'll be able to browse through the docs with Mosaic. LaTeX2HTML would die horribly trying to eat such a monster document. I don't have TeX or LaTeX installed at all, and I don't want to.

Indeed. Use noweave -html, and use HTML markup, not latex markup, in your documentation chunks.

I do want to have reasonable hypertext cross-links between [documents] I've given the matter some thought, and what seems to me the best solution is...

Two questions before you get into solutions: 1) Do you want cross-links for chunks, or only for identifiers? (My religious preference is that chunks should have file scope, therefore no cross-links for chunks.) 2) Where are you going to get the cross-reference information for identifiers? It is hard to get accurate info for C++.

I'd like to see a solution simple enough and general enough that it could be applied to latex or frame or xhdvi documents, not just html documents. Moreover, it ought to be possible to make the solution fit into the nodefs/noweave -indexfrom/noindex framework.

I'm hampered by the whole LaTeX2HTMLmess

Don't touch it. You're not using latex; you have no need. Use HTML, or consider something like linuxdoc-sgml.

What really worries me is that the matter was not even briefly mentioned in any literate programming document I've encountered. Is it because no-one uses literate programming for big projects? Come to think of it, the biggest literate program I know of is TeX, and Knuth is just, eh, out-of-the-ordinary enough to write it as a single .w file, for all I know. This is not an option for a team of five programmers working on a library with >30K of C++ LOC and >100 of C++ classes.

Many people use a large number of separate source files, with all the usual separate-compilation goodies, but then format all the source code in one large document. This is not as awkward as it sounds, especially since there are good facilities for excerpting large documents. What has changed is the advent of HTML, plus browsers that are emphatically not designed to handle large documents.

Am I missing something obvious?

Well, you seem not to see that separate compilation needn't have anything to do with `separate documentation'.

If I wanted this tomorrow, I'd try writing a program to split up a big HTML file on <h1> boundaries, and then the following trick:

```
noweave -html -index -autodefs c *.w > huge.html
splitup huge.html
```

This is much easier than what you propose, because the only part that needs thought is converting intra-document to inter-document hrefs when you split the huge.html. If you want to cheat a little, alter tohtml to spit out something suitable when it sees @file.

From: **Christopher Booth**
Date: 08 Feb 1995

Ben-Kiki Oren writes: Well, I've downloaded noweb and nuweb off the net and looked into both. Both are nice, but probably noweb is the tool for me since both are (alas!) inadequate to my needs, as it is designed to be easily extendible. I'll probably look into the new FWEB release since it supports HTML (which is my target formatter). I'm ``somewhat unhappy'' with LaTeX2HTML.

At any rate, my problem is that both tools have zilch support for separate compilation. Now, I typically place each C++ class in its own set of .hh/.cpp files (with a possible .ii for inlines, etc.). I naturally want to write a single '.w' file per class. [snip]

Seconded. I too have used noweb to write C++ in separate classes like this, and you would gasp to see the kludgy way I have tried to get around this problem. It does seem as though it should be possible to address the issue of referencing other WEB documents without physically including them at the level of WEB code (in whatever dialect :-).

From: **Ben-Kiki Oren**
Date: 16 Feb 1995

I have implemented a patch to noweb-2.6c which supports external cross references between noweb files. I'm not going to post it, though, since Norman is working on adding similar functionality into the `official' noweb release. He is doing an orderly integration into the system, while my patch is a quick-and-dirty hack. He also takes a slightly different approach then mine, so you are better off waiting for him to finish. In case you really need it immediately, I will be happy to send the patch by E-mail. The noweb files should be compatible with Norman's enhancement, or could become so using a trivial filter.

Clueless literate programming questions are welcome

From: **Norman Ramsey**
Date: 01 Feb 1995

Some people may have misunderstood an earlier post of mine to be hostile to people who are clueless newbies to literate programming. False, not so. Clueless and all other questions about literate programming are welcome in this newsgroup. The people who are annoying us are the ones who keep asking DOS/Windows programming questions and other things that have nothing to do with literate programming.

From: **Ryan Davis**
Date: 02 Feb 1995

Norman Ramsey writes: Some people may have misunderstood an earlier post of mine to be hostile to people who are clueless newbies to literate programming. False, not so. Clueless and all other questions about literate programming are welcome in this newsgroup. The people who are annoying us are the ones who keep asking DOS/Windows programming questions and other things that have nothing to do with literate programming.

Good... Then I have a couple of questions: 1) Is there a FAQ? I normally read a newsgroup for more than a month before I start to question this, so... I haven't seen one posted yet... 2) Is Literate programming just a means of pretty printing and automating documentation? 3) Is there any implementations of WEB that are macintosh compatible, OR that are machine independant and will run with PERL or some other widely available language? 4) I've noticed several WEB thingies are C, C++, Pascal... literate(?) but never have I seen the word smalltalk... anyone? Hows that?

From: **Patrick McPhee**
Date: 04 Feb 1995

Ryan Davis writes: 1) Is there a FAQ?

There is. The last one I have was dated May 27, so perhaps it's no longer maintained (or perhaps I just haven't saved it for a while). Try rtfm.mit.edu.

2) Is Literate programming just a means of pretty printing and automating documentation?

No. At least in my view, literate programming is a mechanism for writing programs which are relatively easy to maintain. Apart from encouraging verbose comments, literate programming places emphasis on explaining how and why a program was written the way it was, and it provdes a mechanism for laying the program out in a way which will best aid understanding. A potential side-benefit of this is that the literate programmer has to think twice about everything, which can help uncover problems before they get into production systems and lead to the end of civilisation.

3) Is there any implementations of WEB that are macintosh compatible, OR that are machine independant and will run with PERL or some other widely available language?

There's an MPW litprog tool, which I suppose is in the archive at Darmstadt. I don't really know what languages or formatters it supports, and the Mac programmers where I work were not in any way interested in it :(.

4) I've noticed several WEB thingies are C, C++, Pascal... literate(?) but never have I seen the word smalltalk... anyone?

There are a few packages that don't worry about the source code. FunnelWeb, noweb and nuweb come to mind.

From: **Norman Ramsey**
Date: 06 Feb 1995

Ryan Davis writes: 3) Is there any implementations of WEB that are macintosh compatible, OR that are machine independant and will run with PERL or some other widely available language?

noweb, nuweb, and Funnelweb will let you write programs in any language. I think nuweb just works on Macs. Getting noweb installed on a Mac would be quite a chore.

From: **Eitan Gurari**
Date: 07 Feb 1995

Ryan Davis writes: 3) Is there any implementations of WEB that are macintosh compatible, OR that are machine independant and will run with PERL or some other widely available language?

Norman Ramsey writes: noweb, nuweb, and Funnelweb will let you write programs in any language. I think nuweb just works on Macs. Getting noweb installed on a Mac would be quite a chore.

(Al)ProTex provides another alternative, if you have (La)TeX on your system. No installation is required besides ftp'ing two files.

Ryan Davis writes: 4) I've noticed several WEB thingies are C, C++, Pascal... literate(?) but never have I seen the word smalltalk... anyone?

Trygve Reenskaug and Anne L. Skaar. An Environment for Literate Smalltalk Programming. Proceedings of the 1989 Conference on Object-Oriented Programming: Systems, Languages, and Applications, ACM SIGPLAN Notices 24:10 (1989), 337--345.

From: **Joachim Schrod**
Date: 07 Feb 1995

Ryan Davis writes: 3) Is there any implementations of WEB that are macintosh compatible, OR that are machine independant and will run with PERL or some other widely available language? Patrick McPhee writes: There's an MPW litprog tool, which I suppose is in the archive at Darmstadt.

[I haven't seen the original posting, so I'm responding here.] Currently available for Macs (excerpt from the README) at the LitProg Archive (see below) in machines/mac/. I welcome more contributions.

```
CWEB                - port of CWEB
Impact              - CTANGLE with Apple-events
LPW                 - Literate Programming Workshop
                      ATTENTION: This is shareware!  [50 US-$]
*SpiderWEB*         - port of SpiderWEB, for MPW
```

There are no language independent tools that you can use with Perl, though. My bet is that your best try is to port nuweb -- and send me the result for incorporation in the Archive :-).

From: **David Kastrup**
Date: 14 Feb 1995

Siu-ki Wong writes: What is the purpose of literate programming? I am a newie to this news group, so I don't know if this question has been asked thousands times before. If yes, please excuse for the redundancy. If I understand correctly, in this paradigm of programming, we can have a single source for programs and documentations. Yet, why should we do that in this way? Is there any advantages in doing so? What is its impact on the software development and maintenance? Thanks.

It's the program documentation which sits in the same file, not (necessarily) the user's documentation. The purpose is clear: by having the code and code comments appear at the same place, it is psychologically more annoying to have them disagree, so docs and code tend to be more in sync. Also, being able to explain each code part as a separete entity, even if it is not a function, is a definite plus. Especially for programs for mathematical purposes, it is often handy to be able to use all the capabilities for formulas your typesetting system supports in comments. But I'm babbling. See the FAQ which is posted on a frequent basis here.

From: **Eitan Gurari**
Date: 11 Feb 1995

The following example illustrates what it takes to compile noweb files with protex (for any tab-free program on any platform that supports (la)tex). No need to add protex to the archive, it is already there ;-).

```
------------------ example of a latex file ---------------------

% Requires LaTeX, (Al)ProTex,  the file `noweb.sty' from
% bellcore.com : pub/norman/noweb/tex/         and the following files from
% bellcore.com : pub/norman/noweb/examples/
%
%    breakmodel.nw      dag.nw        graphs.nw     mipscoder.nw
%    multicol.sty       primes.nw     scanner.nw    test.nw
%    tree.nw            wc.bbl        wc.nw         README
%
% Changes needed within the `.nw' files
%
% 1. Insert `@' after each chunk that is not delimited with such a character
% 2. Replace `@<<' with `<@<'
% 3. Clean ``[[...]]'' from special characters that differ from ^ and _

\documentstyle{book}

%-----------  ProTeX adjusted to accept noweb inputs -------

\input ProTex.sty    \catcode`\^^M=13  \def\CodeDel{{=}{
                                       @}}               \catcode`\^^M=5
\AlProTex{any,<<<>>>,title,list,ClearCode,_^,ShowFile,[]}

\let\oldModifyAppendCode=\ModifyAppendCode
\let\oldModifyOutputCode=\ModifyOutputCode
\def\ModifyAppendCode#1{\oldModifyAppendCode{#1}\catcode`\<=13
                                                \catcode`\\=12\relax}
\def\alprotextitle#1#2>>{\egroup \<#2\>}
\def\alprotextitleA#1#2>>{\<#2\>}
\def\ltsym{<}    \catcode`\<=13
\def<#1{\ifx#1<\bgroup \catcode`\$=12 \catcode`\\=12  \par
    \expandafter\alprotextitle  \else  \expandafter\ltapsym\fi #1}
\def\ltapsym#1{\ltsym\ifx#1@\else\expandafter#1\fi}
\def\ModifyOutputCode{\def<##1{\ifx##1<\expandafter\alprotextitleA
           \else  \expandafter\ltapsym\fi ##1}\oldModifyOutputCode}

%-------------------------------------------------

\textwidth=6.9in
\oddsidemargin=0pt
\evensidemargin=0pt

\begin{document}  \tableofcontents

%---------- noweb style files --------------------

\catcode`\@=11     \catcode`\^=7   \input multicol.sty   \input noweb.sty
\catcode`\@=12     \catcode`\^=13

%-------------- tame  [[...]] --------------------

\def\oldcite{\@tempswafalse \@citex []}   \def\bracket{[}   \catcode`\[=13
\def[{\bracket}
```

```
\def\cite{\@ifnextchar [{\@tempswatrue \@citex }{\oldcite}}

%---------------- README ------------------------

\chapter{README}  \ShowFile{README}

       \let\oldinput=\input
       \def\input#1.#2 {\chapter{#1}\openin15=#1.#2
          \ifeof15 \closein15
             \immediate\write16{-----File ``#1.#2'' is missing-----}
          \else \oldinput #1.#2 \fi}

%---------------- breakmodel --------------------

\input breakmodel.nw                                  \ClearCode

%----------------- dag --------------------------

\input dag.nw
<<dag>>=
<<*>>
@                           \OutputCode\<dag\>          \ClearCode

%----------------- graphs ------------------------

                       \begingroup \catcode`\#=12 \catcode`\&=12
\input graphs.nw       \endgroup                            \ClearCode

%----------------- mipscoder ---------------------

\input mipscoder.nw
<<mipscoder>>=
<<*>>
@                           \OutputCode\<mipscoder\>    \ClearCode

%----------------- primes ------------------------

\input primes.nw
<<primes>>=
<<*>>
@                           \OutputCode\<primes\>       \ClearCode

%----------------- scanner -----------------------

\input scanner.nw           \OutputCode\<parser\>
                            \OutputCode\<lexer\>        \ClearCode

%----------------- test --------------------------

\input test.nw
<<test>>=
<<*>>
@                           \OutputCode\<test\>         \ClearCode

%----------------- tree --------------------------

\input tree.nw
<<tree>>=
<<*>>
@                           \OutputCode\<tree\>         \ClearCode

%----------------- wc ----------------------------

\input wc.nw
<<wc>>=
<<*>>
@                           \OutputCode\<wc\>           \ClearCode
```

```
\end{document}
```

Spidery WEB for Perl?

From: **Diego Zamboni**
Date: 01 Feb 1995

Has somebody done something with Spidery Web for programming in perl? Or does anybody know of some literate programming tool apt for programming in perl? Thanks! Please answer by email if possible, I'll summarize.

From: **Colman Reilly**
Date: 01 Feb 1995

I use noweb, which is language independent, and which can handle HTML being the formating language in use.

From: **Matthias Neeracher**
Date: 02 Feb 1995

I'd recommend noweb. Perl syntax is quite complex to parse, so you'd probably spend more time fiddling with pretty-printing than programming (Of course, the fact that this argument is mostly applicable to *every* programming language is exactly what made Norman Ramsey write noweb based on his experiences with Spidery WEB - or that's the impression I got from his papers).

From: **Marc van Leeuwen**
Date: 03 Feb 1995

*Matthias Neeracher writes: I'd recommend noweb. Perl syntax is quite complex to parse, so you'd probably spend more time fiddling with pretty-printing than programming (Of course, the fact that this argument is mostly applicable to *every* programming language is exactly what made Norman Ramsey write noweb based on his experiences with Spidery WEB - or that's the impression I got from his papers).*

You may be right for Perl, but I disagree for the claim about every programming language. Just recently I wrote a spidery web for the GAP (computer algebra) language, which took me no more than two days, including the installation of Spidery WEB itself (that was the hardest part, it comes with the most horrid makefile I have ever had to look at), and playing around a bit with various layout styles; it works like a charm. It all depends very much on the particular language you are using. If your language has a regular and context-independent syntax, and no lexical rules that deviate from the customary ones, then Spidery WEB is fine, depite what its author might say about it. If your language is otherwise, better forget about pretty-printing (as an extreme case consider TeX; I don't think anybody even has any clear idea of what pretty-printed TeX source code would have to look like). Just as an indication, in my opinion Pascal would be fine, C is on the border, and C++ would be over the border (too much context dependence), at least, if you are serious about the full langauge and quality pretty printing. Hand written WEB systems like CWEB and FWEB can provide better results for tricky languages, but if you have to write such a system yourself, this is of course quite an investment. Nevertheless I strongly disagree with people saying that pretty-printing is not worth the effort for anyone in general; this depends very much on ones particular situation and purpose. In some cases pretty-printing may even be the main reason to opt for literate programming.

From: **Matthias Neeracher**
Date: 06 Feb 1995

Marc van Leeuwen writes: You may be right for Perl, but I disagree for the claim about every programming language. Just recently I wrote a spidery web for the GAP (computer algebra) language, which took me no more than two days, including the installation of Spidery WEB itself (that was the hardest part, it comes with the most horrid makefile I have ever had to look at), and playing around a bit with various layout styles; it works like a charm.

I was not trying to imply that Spidery WEB doesn't do its job.

It all depends very much on the particular language you are using. [...] Just as an indication, in my opinion Pascal would be fine, C is on the border, and C++ would be over the border (too much context dependence), at least, if you are serious about the full langauge and quality pretty printing.

Yes, that agrees quite well with my (limited) personal experience. But even when the grammar works well, users of prettyprinting Webs still have to provide formatting hints like @; or @#.

Nevertheless I strongly disagree with people saying that pretty-printing is not worth the effort for anyone in general; this depends very much on ones particular situation and purpose. In some cases pretty-printing may even be the main reason to opt for literate programming.

What I like about noweb is that it doesn't *totally* close the door for all future prettyprinting. It always leaves at least a theoretical possibility to add a prettyprinter to the system. This helps to stop worrying about prettyprinting while programming.

From: **Joachim Schrod**
Date: 07 Feb 1995

Matthias Neeracher writes: Yes, that agrees quite well with my (limited) personal experience. But even when the grammar works well, users of prettyprinting Webs still have to provide formatting hints like @; or @#.

IMO that's because the prettyprinting grammars are so bad. In particular: 1) Most often one has to use `@;' after refinements. That's not needed if the grammar recognizes a rule like `refinement LINE_END -> statement', i.e., if a refinement at the end of the line implies a `@;' behind. 2) Most grammars try to be too cute anyhow. If we look at the success of font-lock or hilit in the Emacs world, we can see that users want prettyprinting, but they want only a limited amount of it.

IMO, the prettyprinter shall not mangle their input and make it unreadable (e.g., CWEB would do this all the time with my programs...). Instead it should augment the input with typographic information. In the moment, when a prettyprinter takes the input code structure created by the author as a very serious hint for the output, users don't flame the system any more. Empower the user! I agree with the final comment on noweb, though. Did anybody write a converter for CWEB -> noweb?

From: **Norman Ramsey**
Date: 08 Feb 1995

*Matthias Neeracher writes: What I like about noweb is that it doesn't *totally* close the door for all future prettyprinting. It always leaves at least a theoretical possibility to add a prettyprinter to the system. This helps to stop worrying about prettyprinting while programming.*

Adding prettyprinters to noweb is more than just a possibility. Kostas Oikonomou has done it for Icon, Turing, and some other languages. Kaelin Colclasure did one for C++. These prettyprinters are included in the noweb distribution.

From: **Marc van Leeuwen**
Date: 14 Feb 1995

Matthias Neeracher writes: Yes, that agrees quite well with my (limited) personal experience. But even when the grammar works well, users of prettyprinting Webs still have to provide formatting hints like @; or @#.

Joachim Schrod writes: IMO that's because the prettyprinting grammars are so bad.

I can agree with that only partly. To begin with, `@#' just adds a bit of vertical space, a beautification that is never really essential, so nobody should complain about having to insert it; it is just like your average TeX layout control code that lets you improve the output slightly if you feel like it. On the other hand, `@;' is quite different, as its omission may frustrate parsing completely. However, if the prettyprinting grammar is good, one should need `@;' only in connection with tricky macros, where the superficial syntax (i.e., viewing the macro simply as a function call) differs from the actual syntax after expansion of the macro. (Somewhat ironically, in the original WEB for Pascal, `@;' is only required because of the macro mechanism that WEB itself had added.) As cases that are superficially unsyntactical look peculiar to the human eye (usually a semicolon seems to be lacking), programmers will probably be aware of them anyway, and it should not be too much to ask them to type `@;' to make the code look superficially correct; all this assuming a good prettyprinting grammar.

In particular: Most often one has to use `@;' after refinements. That's not needed if the grammar recognizes a rule like `refinement LINE_END -> statement', i.e., if a refinement at the end of the line implies a `@;' behind.

Here you must be referring to CWEB or something derived from it, because IMO the real reason for this problem has nothing to do for

WEB systems in general (e.g., it does not apply to Pascal WEB). Rather, it is caused by an error made in converting WEB to CWEB that was never corrected, namely a failure to recognise the fundamentally different roles of semicolons in Pascal and C. In Pascal semicolons separate statements, whether simple (e.g., assignments) or compound, so if any such statement is followed sequentially by another one, a semicolon is required after the first; the requirement still holds if the first statement is a refinement. In other words, in Pascal one has `refinement -> statement' always, without having to consider LINE_END, and without having to use `@;' (either a real semicolon is required after it, or nothing is, e.g., when ELSE follows). In C however, no symbol separates successive statements, but things like assignments are expressions rather that statements, and they only become statements by incorporating a following semicolon. Since refinements almost always stand for (compound) statements, they should be given that category; however in (Levy/Knuth) CWEB they are given the same category as assignments, which is `expression'. This forces one to write a semicolon after each refinement, even though that really becomes a spurious empty statement; if such an empty statement would mess up the real syntax (the compiler would fail) then one has to write `@;' instead. In CWEBx where the category of refinements is `statement', this problem simply does not exist, and you never write a semicolon after a refinement.

I find the solution you suggest a bit more dubious. Not that it would make much difference (in the restricted form you suggest), since in practice refinements always come at the end of a line, so that they are always considered to be statements, which is just what I suggested. But from a more fundamental point of view, this solution implies that line ends will be communicated to the parsing mechanism, supplying it with information that the user did not explicitly intend to be sent. Sometimes a line end will indicate something relevant, like the end of a statement, but that information can also be derived from the programming tokens themselves. On the other hand it will often also indicate something completely irrelevant, like the fact that I have approached the right margin of my favorite text window size (which is quite independent of the right margin of the typeset page). In automatic formatting one should avoid interpreting layout features of the source text unless they are so special that the user can always avoid supplying such features inadvertently[1] (e.g., one could imagine blank lines in the program code being automatically converted to `@#', but even there some programmers may feel that this cramps their input style). It is one of the strong points of TeX that one can format the source text quite freely without having to worry about any effect on the printed output; there is one exception, namely spaces, which are sometimes intended to appear in print and sometimes not, and this is immediately one of the most problematic areas of TeX (too many rules about when to ignore spaces and when not).

[1] I gather from its documentation that in noweb chunk names always have to fit on a single line. If this is true, I wonder how people can cope with a restriction that is so contrary to the spirit of literate programming. If forced to, I think I would choose to use lines that are twice or more the width of my window, but it would not cease to frustrate me.

2) Most grammars try to be too cute anyhow. If we look at the success of font-lock or hilit in the Emacs world, we can see that users want prettyprinting, but they want only a limited amount of it.

I don't think these features of the crude-but-interactive world of text editing have very much to do with prettyprinting for typeset documents. It may very well be that what people really want is a WYSIWYG literate programming tool with all kinds of interactive bells and whistles (hypertext at the very least), and I will certainly not deny them their pleasure, but I think this is not the true goal of literate programming as it was conceived by Knuth (which is I think writing and publishing the best possible documentation for programs that are worth it, in a form that tries to make it as accessible as possible to other human beings).

IMO, the prettyprinter shall not mangle their input and make it unreadable (e.g., CWEB would do this all the time with my programs...). Instead it should augment the input with typographic information. In the moment, when a prettyprinter takes the input code structure created by the author as a very serious hint for the output, users don't flame the system any more. Empower the user!

I already explained why I disagree here. Would you say the same about TeX, i.e., that it should augment the typographic information of the source text, taking the input code structure created by the author as a very serious hint for the output? I am very glad that it does not (except in very limited and well defined ways, like treating blank lines as paragraph separators). I would be very much interested in examples where CWEB would really make programs unreadable. I can imagine several scenarios: (1) The input causes the parser to choke, resulting in horrendous output. (2) CWEB is being used in an inappropriate way, e.g., huge block comments. (3) The input style of the programmer is systematic, but differs radically from the (equally systematic) output style of CWEB. (4) The programmer is using a very ad-hoc style, giving visual clues in ways that the prettyprinter is unable to reproduce.

I think case (1) is the most common (it is the only one that would support your strong wording), but such cases are only serious when it is impossible to modify the source with some hints so that the parser will succeed. You would not be justified either in complaining about your programs running incorrectly when they still contain syntax errors. I consider the property of being silent by default about such syntactic disasters a serious defect of traditional WEB systems, but it is easy to remedy. The solution for case (2) is obvious, and in cases (3) and (4) the unreadability is really a subjective matter. I don't expect that cases (2) and (4) occur very frequently (perhaps I am being too optimistic here). As regards (3), it would be a good thing however if any popular layout style could be selected as an option to the prettyprinter (just like LaTeX allows selection of a document style independently from its contents); in CWEBx a modest attempt is made in this direction. This approach would makes the program much more an abstract document, and independent of the particular preferences of the programmer, unlike your suggestion, which does just the opposite.

From: **Norman Ramsey**
Date: 14 Feb 1995

Joachim Schrod writes: IMO, the prettyprinter shall not mangle their input and make it unreadable (e.g., CWEB would do this all the time with my programs...). Instead it should augment the input with typographic information.

The prettyprinters in the noweb contrib directory have this property; they change the ``look'' of keywords, identifiers, and operators without changing their positions on the page.

Did anybody write a converter for CWEB -> noweb?

I wrote one years ago for spidery web, which at the time was fairly close to CWEB. It was either a monster sed script or a monster sam script, and I'm not sure if I can find it. It would probably be easier just to modify cweave to emit the representation described in the noweb hacker's guide.

From: **Brian Stuart**
Date: 16 Feb 1995

Marc van Leeuwen writes: Here you must be referring to CWEB or something derived from it, because IMO the real reason for this problem has nothing to do for WEB systems in general (e.g., it does not apply to Pascal WEB). Rather, it is caused by an error made in converting WEB to CWEB that was never corrected, namely a failure to recognise the fundamentally different roles of semicolons in Pascal and C. In Pascal semicolons separate statements, whether simple (e.g., assignments) or compound, so if any such statement is followed sequentially by another one, a semicolon is required after the first; the requirement still holds if the first statement is a refinement. In other words, in Pascal one has `refinement -> statement' always, without having to consider LINE_END, and without having to use `@;' (either a real semicolon is required after it, or nothing is, e.g., when ELSE follows). In C however, no symbol separates successive statements, but things like assignments are expressions rather that statements, and they only become statements by incorporating a following semicolon. Since refinements almost always stand for (compound) statements, they should be given that category; however in (Levy/Knuth) CWEB they are given the same category as assignments, which is `expression'. This forces one to write a semicolon after each refinement, even though that really becomes a spurious empty statement; if such an empty statement would mess up the real syntax (the compiler would fail) then one has to write `@;' instead. In CWEBx where the category of refinements is `statement', this problem simply does not exist, and you never write a semicolon after a refinement.

Just to play devil's advocate and because I have used it this way a few times, how do you handle a case where a fragment is indeed meant to replace an expression rather than a group of statements? (@+ ?) For example (albeit contrived):

```
   for( <Each element in the list> ) {
      ...
   }
```

where

```
   <Each element in the list> =
      p = head; p != NULL; p = p->next
```

or the like. It seems to me that this can make such very common idioms very readable to the non-experienced such as students and professionals new to the language. It's also a good place to isolate and discuss assumptions regarding the list like whether or not sentinals are used and the like. Along those lines, if I have many such loops, I can reuse <Each element in the list> many times and if the assumptions of the list change, I only need to find and change this one fragment definition. Of course, I don't use fragments this way nearly as often as I do to replace statements so I would tend to agree that the default behavior should be based on the usage as statements.

Are verbose comments appropriate?

From: **Mel O Cinneide**
Date: 12 Feb 1995

Patrick McPhee writes: No. At least in my view, literate programming is a mechanism for writing programs which are relatively easy to maintain. Apart from encouraging verbose comments, literate programming places emphasis on explaining how

Are verbose comments appropriate? Verbose comments are 1) tedious to read 2) spoil the visual structure of the code 3) are seldom

updated by maintainence programmers and so aren't reliable anyway! I always encourage students to write short, terse comments. Well-written code is often self-explanatory and the complexity involved understanding the larger system is better managed by studying the design documentation. I'd agree with 'plenty of appropriate, terse' comments!

From: **David Kastrup**
Date: 14 Feb 1995

Mel O Cinneide writes: Are verbose comments appropriate? Verbose comments are 1) tedious to read 2) spoil the visual structure of the code 3) are seldom updated by maintainence programmers and so aren't reliable anyway!

Agreed, in part. However, using a literate programming tool, comment sections and code are presented as typographically different entities, so that it is very easy to study just the code, if you do not care for the comments. The comments do not disrupt the code, it is rather so, that the code exemplifies the expos'e made by the "comments".

I always encourage students to write short, terse comments. Well-written code is often self-explanatory and the complexity involved understanding the larger system is better managed by studying the design documentation.

And the design documentation can be arranged very well between code chunks, in order to explain how they are interlinked. I'd suggest that you get a few example files from some literate programing tool in printable form (say, a finished .dvi-file), and take a look at it. Also, you might consider both tools which are prettyprinting and those which are not. I tend to find the latter class more appealing to me, and guessing from the comments you made, it might be you'd do so as well.

From: **pettengill**
Date: 14 Feb 1995

Mel O Cinneide writes: Are verbose comments appropriate? Verbose comments are 1) tedious to read 2) spoil the visual structure of the code 3) are seldom updated by maintainence programmers and so aren't reliable anyway! I always encourage students to write short, terse comments. Well-written code is often self-explanatory and the complexity involved understanding the larger system is better managed by studying the design documentation.

Terse and verbose are in the eye of the beholder. Most novels could be shortened to a dozen pages, and with a library of generic plots could be shortened to a page by simply referencing the plots and subplots by number. However, would one really appreciate these novels more by referring to the notes that indicate the choice and interplay or plot lines?

Don Knuth has merged the design documentation and code into one with WEB. While it is certainly true that code changes are often not reflected in the module comments, it is nearly universally true that design documents NEVER represent the code at any point in time in the life of a project. Unless you are dealing TeX or Metafont or SGB where reading the design and reading the code are one and the same.

From: **Patrick McPhee**
Date: 15 Feb 1995

Mel O Cinneide writes: Are verbose comments appropriate? Verbose comments are 1) tedious to read 2) spoil the visual structure of the code 3) are seldom updated by maintainence programmers and so aren't reliable anyway!

Of course, in a literate program, the code interupts the visual structure of the comments. The question the comments should be answering is `what is this code trying to do?'. It isn't always clear, and sometimes you wonder why something was done the way it was done. I've spent a lot of time picking up other people's programs, and I very often want to ask them what they were trying to achieve with it. Short comments, or even short, terse comments, are not usually enlightening: "x = 0; /* assign 0 to x */" is the classic example of a completely useless, but short, piece of documentation.

I always encourage students to write short, terse comments. Well-written code is often self-explanatory and the complexity involved understanding the larger system is better managed by studying the design documentation.

Ah -- but there is no design documentation. In real world systems, written by professional programmers, design tends to be at a very high level, and there's nothing to describe the way some little piece of the puzzle has been put together. In a literate program, at least in theory, the design documentation is stored in the program source file. It's just as tedious to read there as it would be somewhere else, but it's right there, where it's handy if you're trying to figure out what the last SOB that touched this code was trying to do. Of course, the point I was trying to make before (I think I've edited it out of this response) was that by encouraging you to write a

description of what you're trying to do (aka a verbose comment), literate programming encourages you to think about what you're trying to do, which is likely to reduce the need for maintenance in the future.

So, I would argue the opposite to `plenty of terse comments' -- give me one cogent explanation of what you're trying to do, and I can read your code to see what you actually did do. There isn't much you can tell me in a terse comment that's likely to be of any help.

From: **Niall Teasdale**
Date: 15 Feb 1995

Mel O Cinneide writes: Are verbose comments appropriate? Verbose comments are 1) tedious to read 2) spoil the visual structure of the code 3) are seldom updated by maintainence programmers and so aren't reliable anyway! I always encourage students to write short, terse comments. Well-written code is often self-explanatory and the complexity involved understanding the larger system is better managed by studying the design documentation.

Interesting. You know maintenance programmers who don't update the comments, but can be bothered to update the design documentation? One of the best things about literate programming styles is the ability to keep some or all of the design documentation inline with the code. It has the potential of reminding programmers that the design document is there to be updated. I'm not saying it will always work, but it maybe has a better chance.

From: **Thorbjoern Andersen**
Date: 16 Feb 1995

Mel O Cinneide writes: I always encourage students to write short, terse comments. Well-written code is often self-explanatory and the complexity involved understanding the larger system is better managed by studying the design documentation.

My major incentive for using a literate programming tool is the ability to *easily* plug in a descriptive graph of e.g. a data structure. Pointer usage is *much* easier if you have the data structure easily laidout by a good drawing (NOT ascii). Just because you have the opportunity to use a lot of words, don't mean that you *HAVE* to write a lot.

I'd agree with 'plenty of appropriate, terse' comments!

Same here - I just want to have a choice. A good explanation of a complex data structure is hard to put in comments around the code. This usually goes in a block of text anyway, so why not typeset it properly; then the terse comments relate to the full explanation. Besides, too many comments can make code hard to read, and maintain.

From: **Mel O Cinneide**
Date: 22 Feb 1995

Patrick McPhee writes: Of course, in a literate program, the code interupts the visual structure of the comments.

Isn't this a damning remark about literate programming? The code, which is a precise statement of what is being done, takes second place to the informal description by the programmer of what they wanted to do.

The question the comments should be answering is `what is this code trying to do?'. It isn't always clear, and sometimes you wonder why something was done the way it was done. I've spent a lot of time picking up other people's programs, and I very often want to ask them what they were trying to achieve with it. Short comments, or even short, terse comments, are not usually enlightening: "x = 0; / assign 0 to x */" is the classic example of a completely useless, but short, piece of documentation.*

This a strawman if ever I saw one! Do we need to say that comments which give an informal description of the semantics of program constructs aren't useful?

Ah -- but there is no design documentation. In real world systems, written by professional programmers, design tends to be at a very high level, and there's nothing to describe the way some little piece of the puzzle has been put together.

What are you getting at here? There's no design documentation so we need lots of comments? I completely agree with you that puzzling pieces of code must be commented to explain what's going on, but these descriptions should be terse and to-the-point, not verbose. As Meyers puts it, a reader with literary inclinations would rather read Henry James.

In a literate program, at least in theory, the design documentation is stored in the program source file. It's just as tedious to read there as it would be somewhere else, but it's right there, where it's handy if you're trying to figure out what the last SOB that touched this code was trying to do. Of course, the point I was trying to make before (I think I've edited it out of this response) was that by encouraging you to write a description of what you're trying to do (aka a verbose comment), literate programming encourages you to think about what you're trying to do, which is likely to reduce the need for maintenance in the future.

I agree with the aim: you should of course think about what you do when you program. However, attempting to clarify what you're doing by verbose text is going about it the wrong way, IMO.

From: **Andrew Mauer**
Date: 22 Feb 1995

Mel O Cinneide writes: Isn't this a damning remark about literate programming? The code, which is a precise statement of what is being done, takes second place to the informal description by the programmer of what they wanted to do.

I do not believe so. The process of realizing the (abstract) goals of your program can and do obscure a useful (usually high-level) understanding of what the goals are. It is true that the code is a precise statement of what is being done is true, but this focuses on the implementation rather than the design. Although the analogy has its flaws, I think this is similar to saying that assembly code provides a more precise description than a C++ program... (you might get confused because it might not be obvious which virtual function was being called, for instance).

I think that one of the tenants of literate programming is that understanding the existing code (in order to change it) is the most difficult part of the maintenance process. To understand the module that you are changing, you will want the high-level description of most parts (so you can skip over them because you know they are irrelevant), and only read the code in a small section.

I completely agree with you that puzzling pieces of code must be commented to explain what's going on, but these descriptions should be terse and to-the-point, not verbose. As Meyers puts it, a reader with literary inclinations would rather read Henry James.

For me, literate programming makes the documentation/comments reflect the higher level structure, rather than the immediacy of (x=0). Most maintenance needs just a high level understanding of most program parts.

From: **Felix Gaertner**
Date: 24 Feb 1995

Andrew Mauer writes: It is true that the code is a precise statement of what is being done is true, but this focuses on the implementation rather than the design. Although the analogy has its flaws, I think this is similar to saying that assembly code provides a more precise description than a C++ program... (you might get confused because it might not be obvious which virtual function was being called, for instance).

The code is a precise statement of what is being done, sure enough. But is this also the thing that the programmer intended? I think it is one of the benefits of the literate programming paradigm that you are encouraged to first say (informally) in prose what you want to do, and then express (formally) what is done. This just as an aside.

From: **Patrick McPhee**
Date: 24 Feb 1995

Patrick McPhee writes: Of course, in a literate program, the code interupts the visual structure of the comments.

Mel O Cinneide replies: Isn't this a damning remark about literate programming?

It could be. It is, however, and in my opinion, the essence of literate programming.

The code, which is a precise statement of what is being done, takes second place to the informal description by the programmer of what they wanted to do.

The code doesn't work. The code is a precise statement of how to do the wrong thing. But why? What's wrong? We can look at the

design docs, and they'll tell us what this piece of code is supposed to do. This is how we know it doesn't work -- it doesn't do what the design docs say it's supposed to do. The problem is that the original programmer wasn't very good at his job, and he's been fired. It's not obvious from his code what approach he was taking to the problem, and it's not easy to see what's gone wrong if you don't know what he was trying to do. Sometimes there's no choice but to start over, because it's difficult to see how the different pieces of the code fit together.

The problem with this argument is that a really useless programmer is not very likely to write a coherent explanation of what he's doing, anyway. On the other hand, if you force this bad programmer to write a description of the operation of each piece of code he writes, you may get him to recognise that some pieces are convoluted messes which should be rewritten before someone gets hurt. I suggest that this big picture effect won't be introduced by localised comments (which we've been calling terse), since such comments usually take into account only a few instructions of code.

Patrick McPhee writes: Short comments, or even short, terse comments, are not usually enlightening: "x = 0; / assign 0 to x */" is the classic example of a completely useless, but short, piece of documentation.*

Mel O Cinneide replies: This a strawman if ever I saw one!

No, it's an extreme example. My assertion is that it's difficult to say the sorts of things I like to see in comments in only a few words, and I'm not usually interested in the programmer's opinion of things at an extremely localised level. I've seen things like this: "x = 0; /* the x factor has to be reset */" and you have to be in tune with what's going on to understand this. Why does it have to be reset? What is the x factor anyway? Just the statement in the code gets me asking those questions, and the comment doesn't do anything for me. If it's really difficult to understand why x is being reset at this point, I would personally rather see something like this: `About half way through we reset |x| because Lipmann ({\it The X Factor in Programming}, pp 27--43) says it would be a terrible disaster if we didn't.' [ok -- I wouldn't rather see that, it's a parody going the other way. I'd like to see obscure things put into perspective].

Mel O Cinneide writes: [if you want the big picture, look in the design docs]

Patrick McPhee replies: [but there are no design docs]

Mel O Cinneide replies: What are you getting at here? There's no design documentation so we need lots of comments?

There are no design docs at a level that helps in maintianing code, so your code should contain enough information to fill this void. Again, this seems to me to be the point of literate programming -- the proper place for a description of how a program should work is in the source code of the program.

I completely agree with you that puzzling pieces of code must be commented to explain what's going on, but these descriptions should be terse and to-the-point, not verbose.

Certainly to the point. Almost never terse. $terse \to cryptic$. (That's terse approaches cryptic). Plus, I want to know about the code that was left out. I like comments that say `I didn't use <obvious approach> because <subtle problem reared its ugly head>'. I don't have a big problem with comments that say `I used this bad approach because the good approach would take too long, and I've too much else to do.' At least you know where you stand.

As Meyers puts it, a reader with literary inclinations would rather read Henry James.

But someone who wants to maintain a piece of software wants to know what the original programmers were trying to do.

From: **Stephen Boyan**
Date: 24 Feb 1995

verbose versus terse (commenting)

FWIW, The text editor WYLBUR had these as mode commands. Hypotactic versus paratactic literate programming styles could be the subject of a thread in this symposium. Examples of literate programs that Faulkner might have written could be compared to ones by Hemingway. That's a nice thing about literate programming. One can write not just comments, but a whole book, or at least a short story, about the program. ;-)

Is this a good style?

From: **Darrell Grainger**
Date: 18 Feb 1995

I have just started reading this newsgroup. Am I right in thinking this newsgroup is about making your code more understandable and maintainable by proper commenting (or documentation)?

If that is what we are talking about here then it is a good style to write my program as comments that say what I want to do then fill in the space between the comments with the appropriate code? This is what I typically do now. I'll format the comments by cutting and pasting just as I would edit an essay or report. Once I feel the entire project is adequately described I start filling in the code that goes with each comment. Some comments will require entire functions or blocks of code where as the more cryptic lines will have a comment for each line of code. Any suggestions for improvement?

From: **Stephen Boyan**
Date: 22 Feb 1995

Your approach to writing the comments first is very LP. Literate programming should offer you better tools for this. Another aspect of literate programming is the psychological. Literate programming is a programming paradigm that combines the source for a compiler-readable and a human-readable (and maintainable) program in one file. Sections of the program can be introduced to the human reader in the psychologically optimal (as you, the author, see that) order; the preprocessor creating the compiler input will rearrange things into the correct programmatic order, but the order of the maintenance document will remain unchanged. This lets the program be created more like a book. Pure commenting would follow the order of the compiler source.

IMO, this emphasis on the psychological aspects of the code is very much woven into the entire concept of literate programming. Pretty printing is partly a psychological thing, and is a major issue in literate programming.

From: **Kimball Thurston**
Date: 22 Feb 1995

I'm a bit of a newbie with this whole literate programming 'thang, but I have a (I think) straight forward request that isn't totally asinine... I would like to start/resume a thread on people's thoughts or experiences with changing a development project already in progress to use a literate programming system. We currently have several design documents (object models, screen docs of what a screen is supposed to do, etc.) for each module we are developing, and then there are the inline comments and documentation. Of course, my team is having problems keep the two synchronized. It seems as if transferring existing work over would be a nightmare, requiring a bit of overhead... My question is do people think that having the documentation contain the code is worth the time to switch over, or should I just wait until the next phase of the project, and start then?

Current status of NUWEB?

From: **Stephen Simmons**
Date: 22 Feb 1995

My first foray into literate programming was to obtain NUWEB 0.87b and port it to a PC. While reading the manual for NUWEB, I decided that there was one aspect of it that I would like to change. In particular, the LaTeX formatting for the WEB source code is rigidly defined in NUWEB's source code. This means that there is no simple way of redefining the visual appearance of NUWEB's LaTeX output without altering the source code and redefining. (to change, for example, 'scrap' to 'section' and use something other than a diamond to mark the end of each section.)

To my mind, it would be more in keeping with the general philosophy of LaTeX (and especially LaTeX2e) if the NUWEB source code just enclosed the WEB file's source code in an environment, and a header file such as NUWEB.STY could be used to change the visual appearance of the LaTeXed output without having to recompile NUWEB. What is the current status of NUWEB? Is Preston Briggs or anyone else developing NUWEB further? Do people who have had more experience with literate programming than me (i.e. more than none!) feel that this would be sensible/worthwhile?

From: **Norman Ramsey**
Date: 24 Feb 1995

I don't want to put words into Preston's mouth, but let's look at the consequences of his decision: 1) The generated .tex files can be

mailed anywhere on the net, and they just work with any vanilla latex installation --- no special macro files needed. 2) nuweb programs look the same anywhere in the world. 3) It's hard for people to tinker with the appearance of the TeX document, so they might spend their time on something more productive, like writing better documentation or building that C++ cross-refernecer I've been wanting :-)

I took a different approach in noweb. the document is represented in an `intermediate form' that is easy to translate into latex or HTML, and I provided a custom macro package for noweb documents. What are the consequences: 1) I had to write a lot of crufty latex code. It's really embarrassing. I cringe when I have to change this code. 2) The latex code has lots of macros and hooks to change the appearance of documents, which makes the system more complicated. 3) Even worse, a bunch of appearance-related options found their way onto the noweave man page, which has reached the embarrassing stage. 4) Noweb is harder to install --- you have to have write permissions on the directory where latex styles and packages live.

Having said all that, noweb may still be the way to go if you want something customizable. You might have better luck starting from Dave Hanson's `noweave.simple' back end than by trying to modify the existing macro package. ``If I ever find time to do it over, I'll do it better..."

---

noweb: proof of the pudding

From: **Ozan Yigit**
Date: 07 Mar 1995

In case you do not already know this: there is now an industrial-strength example of noweb in the form of a compiler book by Fraser and Hanson.[1] the book is nicely organized and cleanly typeset, and the code fragments are mercifully free of typographic junk. Each fragment has directional annotations to help locate the continuations. [most pages have a mini-index to help find identifiers or their explanations, but I do not know if this was generated by noweb] Very good book, and noweb helps.

[1] Christopher Fraser and David Hanson, "A Retargetable C Compiler: Design and Implementation", The Benjamin/Cummins Publishing Co., 1995. ISBN 0-8053-1670-1

From: **Norman Ramsey**
Date: 07 Mar 1995

Ozan Yigit writes: In case you do not already know this: there is now an industrial-strength example of noweb in the form of a compiler book by Fraser and Hanson.[1] the book is nicely organized and cleanly typeset, and the code fragments are mercifully free of typographic junk.

What do you mean by `typographic junk'? does the stuff like: "This definition is continued on pages 3a, 7b, and 99. This code is used on pages 11 and 13." qualify as junk? Hanson and Fraser have a much more compact notation for this information, approximately:

```
     <definition of thing>=                11 13   3a->
```

where the arrow indicates that the definition is continued. Or in other words

each fragment has directional annotations to help locate the continuations.

I'm folding something like their work back into noweb. The question for this forum is: What should the default be? The Hanson/Fraser style is nice for those of us in the know, but it requires more explanation to novices.

[most pages have a mini-index to help find identifiers or their explanations, but I do not know if this was generated by noweb]

noweb provides the raw info to LaTeX, then David Hanson wrote an Icon program to dig in the .aux file and build the mini-indexes.

Meta literate programming

From: **Mark Kramer**
Date: 08 Mar 1995

The question of Frank Niessink about duplicating code, reminds me of a related problem: duplicated documentation. Now I don't mean documenting the same code twice, which I opposed to in a previous message. But the problem of double documentation occurs at a higher level, as well. Often, there is much overlap between the User Manual and the (documentation of the) implementation of the user interface. However, this overlap occurs in the documentation part rather than the code proper.

I have been thinking about the possibility of 'meta literate programming' to solve this problem: consider the user manual as 'code', which has to be 'documented' together with its implementation. So the description of the user interface is derived from a literate program, just like its implementation! Problem: in the user manual you want to use all possibilities of your text processor (wordprocessor or formatter). But, one level higher, that same text processor has to handle the user manual as 'code'. I have not yet figured out a solution. (Presumably, it can be done in TeX? With a WYSIWYG wordprocessor I foresee tremendous problems.)

An approximate solution might be the following: Start from the user manual, and add 'invisible' subsections that contain the implementation and its technical documentation. This can be done with a text processor that supports 'folding'. To print only the user manual 'fold' all other sections, to print the literate program in full 'unfold' them. It might even be possible to 'fold' parts of the user manual that are less relevant to the implementation. One of our students has been experimenting with this 'folding' approach for another purpose. The results look promising, but it turns out to be less simple than I describe it here. BTW: This idea of meta literate programming seems to solve Frank's problem, too. The two 'documentations' of the same code can be derived from the same meta-source.

From: **Tomas Willis**
Date: 08 Mar 1995

Mark Kramer writes: The question of Frank Niessink about duplicating code, reminds me of a related problem: duplicated documentation. Now I don't mean documenting the same code twice, which I opposed to in a previous message. But the problem of double documentation occurs at a higher level, as well. Often, there is much overlap between the User Manual and the (documentation of the) implementation of the user interface. However, this overlap occurs in the documentation part rather than the code proper.

I have been thinking about the possibility of 'meta literate programming' to solve this problem: consider the user manual as 'code', which has to be 'documented' together with its implementation. So the description of the user interface is derived from a literate program, just like its implementation!

I've tried something like this, using nuweb, since nuweb is rather easy and language-independent. The output from the souce includes the various code source files (C, REXX, whatever) plus a user manual source file (TeXinfo, HTML, whatever). A section of the web might be (meta-syntactically>

```
<Comments for Web>
<TeXinfo User Manual Source>
<Program Code>

                _
               /  TeXinfo source -> user documentation
nuweb source --=-- LaTeX web ------> prettyprinted programmer documentation
               \_ C source -------> executable
```

BTW: This idea of meta literate programming seems to solve Frank's problem, too. The two 'documentations' of the same code can be derived from the same meta-source.

This is more fun to write with a folding editor, for sure.

From: **Felix Gaertner**
Date: 09 Mar 1995

Tomas Willis writes: I've tried something like this, using nuweb, since nuweb is rather easy and language-independent. The output from the souce includes the various code source files (C, REXX, whatever) plus a user manual source file (TeXinfo, HTML, whatever). A section of the web might be (meta-syntactically>

```
<Comments for Web>
<TeXinfo User Manual Source>
<Program Code>

                _
               /  TeXinfo source -> user documentation
nuweb source --=-- LaTeX web ------> prettyprinted programmer documentation
               \_ C source -------> executable
```

I think there has been some talk on this topic a few months ago in this newsgroup. The problem is to keep several documents consistent with eachother. The proposed solution that I read from the current (and previous) discussion is to subdivide these documents and put the bits that have to be kept consistent together in a kind of `meta document'.

I think this solution is only practical if the number of documents that you want to handle in parallel is small, like in traditional literate programming where you have source code and its documentation together. When you want to handle a third document at the same time (like a user's manual) this gets a little more complicated as the structure of a user's manual might differ strongly from the source code documentation. If the number of different documents gets even higher it seems to be quite impossible to write such a meta document. I recall a posting in this group about half a year ago (was it by Norman Ramsey?) about a project that tried to write code, code documentation, user manual, reference manual, etc. at the same time. If my memory still works, the results from this project weren't too promising. (I'd be glad about a reference to it, if somebody remembers the details again.)

IMO a lot can be done with the right tools: You need a `canonical representation' of your `meta document' and an editor that manages different views on it. Folding editors are a first step, but the editor must also be able to rearrange the text entirely, if for example you are focussing on the user manual and not on the code and its documentation. Are there systems existing that already have such a functionality? How do other people cope with this problem?

From: **Colman Reilly**
Date: 09 Mar 1995

Mark Kramer writes: The question of Frank Niessink about duplicating code, reminds me of a related problem: duplicated documentation. Now I don't mean documenting the same code twice, which I opposed to in a previous message. But the problem of double documentation occurs at a higher level, as well. Often, there is much overlap between the User Manual and the (documentation of the) implementation of the user interface. However, this overlap occurs in the documentation part rather than the code proper.

I have been thinking about the possibility of 'meta literate programming' to solve this problem: consider the user manual as 'code', which has to be 'documented' together with its implementation. So the description of the user interface is derived from a literate program, just like its implementation! Problem: in the user manual you want to use all possibilities of your text processor (wordprocessor or formatter). But, one level higher, that same text processor has to handle the user manual as 'code'. I have not yet figured out a solution. (Presumably, it can be done in TeX? With a WYSIWYG wordprocessor I foresee tremendous problems.)

Yes. I've done it with TeX on a UNIX system: it's actually just a matter of extracting stuff using sed, so far. I haven't done anything sophisicated yet, so all that happens is that you enclose your user documentation between \begin{user}..\end{user} pairs and it gets extracted and placed in another file to form the user manual. If you remind me, I'll put up an example of what I've done later today.

From: **Joachim Schrod**
Date: 09 Mar 1995

Felix Gaertner writes: I think this solution is only practical if the number of documents that you want to handle in parallel is small, like in traditional literate programming where you have source code and its documentation together. When you want to handle a third document at the same time (like a user's manual) this gets a little more complicated as the structure of a user's manual might differ strongly from the source code documentation. If the number of different documents gets even higher it seems to be quite impossible to write such a meta document. I recall a posting in this group about half a year ago (was it by Norman Ramsey?) about a project that tried to write code, code documentation, user manual, reference manual, etc. at the same time. If my memory still works, the results from this project weren't too promising. (I'd be glad about a reference to it, if somebody remembers the details again.)

Our largest system written with literate programming methods has 20 MB of source. Most modules have many implementations. That's where I learned that it's impracticable to keep the interface specs in the same file as the implementation. The system is an optimising DVI driver (i.e., compiler backend) family for ca. 20 target architectures that runs on 8 different platforms (all Unix variants counts as one platforms there).

The user manual has to be targeted towards architecture and platform specialities. It is a multi-document in itself, i.e., a hypertext where different may be typeset and printed. Its abstraction level and its organisation is (has to be!) completely different from both the design documents & the interface specs & the (multiple!) implementation docs. It is already a maintainance nightmare. (Christine Detig presented the basic paradigms at the EuroTeX conference in Aston.) Intermixing it also with code documentation? Heaven forbid.

This does not mean that its made completely independent from code documentation. We extract automatically user-level information from our literate program sources (in particular, about configuration possibilities and error messages, with possible causes & repairs), that are used to keep appendices up to date. But in the main body of our user manuals it would disturb the narrative flow of information.

From: **Norman Ramsey**
Date: 09 Mar 1995

Mark Kramer writes: The question of Frank Niessink about duplicating code, reminds me of a related problem: duplicated documentation. Now I don't mean documenting the same code twice, which I opposed to in a previous message. But the problem of double documentation occurs at a higher level, as well. Often, there is much overlap between the User Manual and the (documentation of the) implementation of the user interface. However, this overlap occurs in the documentation part rather than the code proper.

I have been thinking about the possibility of 'meta literate programming' to solve this problem: consider the user manual as 'code', which has to be 'documented' together with its implementation. So the description of the user interface is derived from a literate program, just like its implementation!

It's been done, and it's been done multiple ways. One way is to put user documentation into code chunks, then use tangle to extract it. When you weave the source, you elide those chunks. I've used this to good effect with man pages and reference manuals, where the `code chunks' contain troff or TeX. Even if you do this very carefully, you wind up with source code (as opposed to woven code) that is nearly impossible to understand. We need better editing support :-(

The other way it's been done is to use `conditional compilation' in the word processor (e.g, TeX). So, for example, you surround user manual stuff with \ifusermanual...\fi (or one of the nifty conditional environments if you're a latex2e master), and you set the same document different ways depending on those options. (Usually the user manual elides all the code.) This method is *much* less flexible --- you must find a single structure that works for all documents --- but the source code is better structured. I don;t find the game worth the candle, so when I have to do this stuff I fall back on the first method.

From: **Felix Gaertner**
Date: 09 Mar 1995

Joachim Schrod writes: [Writing code documentation and user manual together...] This does not mean that it [the user manual] is made completely independent from code documentation. We extract automatically user-level information from our literate program sources (in particular, about configuration possibilities and error messages, with possible causes & repairs), that are used to eep appendices up to date. But in the main body of our user manuals it would disturb the narrative flow of information.

This is exactly the point I want to make: The documents have different structures and so intermixing code documentation and user manual text doesn't make too much sense, because it disturbs "the narrative flow of information". But it still makes sense to have parts of the texts `connected' in some way. (An intelligent tool could hide the code documentation if you want to edit the user manual for example.)

Joachim's example seems to be quite common: You want error messages to show up in the user manual. But naturally they are part of the program code. Solution one: You copy the messages by hand into the user manual and format them nicely. But then you're stuck if the code changes. It would be nice to be able to reference these code bits from the user manual, so that the text processor automatically copies them into the right spot when you're looking at it. I have seen projects where this `referencing' was done like Joachim presents it: as an automatic extraction process (using sed scripts). I am thinking of a more direct way, i.e. tagging the messages in a certain way (with a label) and referencing the label in the user manual. How did you do it, Joachim?

From: **Joachim Schrod**
Date: 09 Mar 1995

Felix Gaertner writes: Joachim's example seems to be quite common: You want error messages to show up in the user manual. But naturally they are part of the program code. Solution one: You copy the messages by hand into the user manual and format them nicely. But then you're stuck if the code changes. It would be nice to be able to reference these code bits from the user manual, so that the text processor automatically copies them into the right spot when you're looking at it. I have seen projects where this `referencing' was done like Joachim presents it: as an automatic extraction process (using sed scripts). I am thinking of a more direct way, i.e. tagging the messages in a certain way (with a label) and referencing the label in the user manual. How did you do it, Joachim?

We did both: We tagged it in the CWEB source, CTANGLE wrote it to special auxilliary files, and we transformed the tags later by an external sed script (markup for CWEB sources and LaTeX source is different here). It would have been possible to write TeX macros to parse the auxilliary files directly, but the sed script was less work and is more maintainable. I think we have more tools in our box than just *WEB and TeX, we should use them all. Btw, this was my own CWEB (cf. the TUGboat article), not the Knuth/Levy one. Another reason why we never did source distributions of our drivers.

From: **Stephen Boyan**
Date: 11 Mar 1995

double documentation

In the IBM Bookmanager bookshelves there are often twenty or so documents relating to a program product. These are mainframe products, but with the increasing robustness and functionality of micro software and network software this may not be relevant. I don't think creating one source file to create twenty books is currently manageable. However, some kind of configuration management tool might make the set of documentation for the product manageable. I view literate programming as more focused on documenting an individual program for maintenance and enhancement. The double documentation subject is certainly worthy, but I think it belongs more in the configuration management arena.

From: **Eitan Gurari**
Date: 11 Mar 1995

Mark Kramer asks: Presumably, it can be done in TeX? Felix Gaertner replies: I have seen projects where this `referencing' was done like Joachim presents it: as an automatic extraction process (using sed scripts). I am thinking of a more direct way, i.e. tagging the messages in a certain way (with a label) and referencing the label in the user manual.

The example at the end of this note illustrates how the above specifications can be realized in the TeX-based system ProTeX.

1. The `code part' uses the following tagging conventions.

a. External titles for code fragments \<....\>

b. Delimiters for code fragments >>>

c. Internal titles for code fragments `<....`>

2. The `manual part' uses the following tagging conventions.

a. External titles for code fragments \<....\>

b. Delimiters for code fragments ///

c. Internal titles for code fragments @<....@>

3. The `code part' transfers messages to the `manual part' indirectly through an intermediate file.

In the current context, the different tagging conventions enable `code part' to embed code fragments of `manual part' within its code fragments. In more general cases, such conventions allow for hierarchical definitions of literate programs. Families of programs and documents can be defined from a `code part' and a `manual part' by parameterizing them with a common set of tags.

```
-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-

\input ProTex.sty

\begingroup

                            CODE PART

\def\CodeDel{{}{>>>}}
\AlProTex{cd,<<<>>>,title,list,NoShow,`,ShowFile}

\<code 1\>
111111111>>>

\<code 2\>
222222222>>>

\<code 3\>
333333333>>>

\<code\>
  `<code 1`>
  `<code 2`>
  `<code 3`>
>>>
                        \ShowOff
\<doc\>
```

```
   \<doc 1\>
      `<code 1`>
      `<code 3`> ///
   \<doc 2\>
      `<code 1`> ///
>>>
        \OutputCode\<code\>     code.cd:\par  \ShowFile{code.cd}
        \OutputCode[dc]\<doc\>  doc.dc:\par   \ShowFile{doc.dc}

\endgroup    -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-

% \bye       \input ProTex.sty

                              MANUAL PART

\def\CodeDel{{}{///}}
\AlProTex{,<<<>>>,title,@}

\input doc.dc
\ShowCode\<doc 2\>

...bla, bla, bla...

\ShowCode-\<doc 1\>
\bye
```

From: **Kasper Osterbye**
Date: 12 Mar 1995

The issue of how one creates several documents on top of the SAME program; documents with different structure and contents, I believe is important, and I am convinced it is difficult to solve using the existing literate tools. For a while a colleague and I have been experimenting with literate programming based on hypertext, and it seems to promise some way out of this, but at the expense of considerable more complex tools.

The main idea is the following. 1) Program and documentation(s) are stored as a hypertext. For the typical example of C++ and LaTeX, this would mean that each program representing node will typically store something like a function (member or normal function) or a class definition. There will be links from the class definition to member functions. The documentation will typically be arranged with a section per node, and a linking structure that represents the hierarchical structure of the document. 2) The editor is able to present a node, but rather than just showing the links as buttons, some links destination is show inlined. This means that a piece of documentation with a link to source, can be shown inlined its documentation, giving a typical "literate" look.

As the source code and documentation are separated in the internal representation but juxtapositioned in its presentation we get the typical literate advantages, but opens up for the possibility that several documents can be build on top of the same program. A program library could for instance have a literate document and a reference manual.

Regarding the maintenance of two documents, we have been experimenting with letting the editor be set up such that the source code can only be edited when shown inlined in the literate view, but disallowing the prototypes and class definitions to be edited through the literate view, but allowing this sort of code to be edited only through the reference manual.

The underlying mechanisms of the editor can also be set up to provide linearization of the program, providing a tangle, only we need not tangle the program from its documentation, because they are already separated at the internal level. During the actual editing, the typical editor takes care of juxtapositioning the code and document, which ever document we are talking about. To produce paper output, we have recently experimented with a document linearization which produces nuweb. So far this has proven to be a reasonable way of doing it.

Keeping the program separate from its documentation, and in a structured internal representation also allow us to give various outlines of the program. The message here is not that hypertext is a fine way of PRESENTING literate programs on a screen, but a fine way of REPRESENTING programs and documentation internally. One would expect many of the same advantages from using an internal representation of abstract syntax trees, only one has to have a good model for linking documents to code. We have a paper on this in the 94 European Conference on Hypermedia technology. In the Feb. 95 issue of IEEE Trans. on SE, I present the argument that hypertext has interesting properties regarding the PRESENTATION as well.

From: **Dave Murphy**
Date: 14 Mar 1995

I have been thinking about the possibility of 'meta literate programming' to solve this problem: consider the user manual as 'code',

which has to be 'documented' together with its implementation. So the description of the user interface is derived from a literate program, just like its implementation!

Ummmm, IMHO The user manual should be it's own document, also your suggestion - where does it stop? Document the document that documents the user manual? Most people working in real projects haven't time (or money) to provide documents properly as is without adding extra layers of documents to maintain.

I think there has been some talk on this topic a few months ago in this newsgroup. The problem is to keep several documents consistent with each other. The proposed solution that I read from the current (and previous) discussion is to subdivide these documents and put the bits that have to be kept consistent together in a kind of `meta document'.

I think this solution is only practical if the number of documents that you want to handle in parallel is small, like in traditional literate programming here you have source code and its documentation together. When you want to handle a third document at the same time (like a user's manual) this gets a little more complicated as the structure of a user's manual might differ strongly from the source code documentation.

User documents WILL differ enormously from the program, and average users in, say, an accountants office would be none to chuffed to see a pile of 'c' like code to work from rather than a user centered manual. From a user's point of view - they would rather the program was written in a structure to reflect their business needs (or better still, know nothing about the structure.

IMO a lot can be done with the right tools: You need a `canonical representation' of your `meta document' and an editor that manages different views on it. Folding editors are a first step, but the editor must also be able to rearrange the text entirely, if for example you are focussing on the user manual and not on the code and its documentation. Are there systems existing that already have such a functionality? How do other people cope with this problem?

Aren't case tools supposed to handle this? Start with a business problem definition, and then work by stages to the end of a finished system. Of course no case tools really achieve all this in practice but the idea seems sound enough. Define the business processes in a tool/repository - then relate each step of system design to that, then each step of code production to the design. User docs would then be built as set of 'this is how you use the system to perform your tasks' type book, but each set of tasks would already relate to the business processes originaly defined (if the definition was done properly and in sufficient detail). The packaging of the user doc is largely irrelevant as it should pick up the individual components from your repository. All subsequent changes to the system should be as requests from the business to achieve a given aim, which would go through the steps outlined so forcing docs to keep in step. This whole view pre-suposes an elctronic based method, rather than traditional methods merely grafted to partial electronic tools. Oh well, it was a nice dream.

From: **Colman Reilly**
Date: 14 Mar 1995

Joachim Schrod writes: Our largest system written with LitProg methods has 20 MB of source. Most modules have many implementations. That's where I learned that it's impracticable to keep the interface specs in the same file as the implementation. The system is an optimising DVI driver (i.e., compiler backend) family for ca. 20 target architectures that runs on 8 different platforms (all Unix variants counts as one platforms there).

This does not mean that its made completely independent from code documentation. We extract automatically user-level information from our literate program sources (in particular, about configuration possibilities and error messages, with possible causes & repairs), that are used to keep appendices up to date. But in the main body of our user manuals it would disturb the narrative flow of information.

I would tend to agree: containing the entire user manual in the technical documents would seem very difficult. On the other hand, the stuff that changes when one is changing the code should be near the code so that the programmer can see the problem in his docs.

For example: I am working on a reasonably complicated Mathematica package, and I would like to build a technical glossary and a reference manual. I include in my code parts of the reference manual and the glossary information: the reference manual chunks glue together to form a reasonable description of the code, and in the technical docs they get typeset in boxes, giving a description of what this code should do, so that the user reference manual is included in the technical documentation. This allows the programmer to see what the code is meant to do without my needlessly repeating myself.

Not to mention that if I had to maintain user and technical documentation seperately I'd suffer complete meltdown. Ideally I'd get around to having the system act in a more intelligent fashion, allowing me to rearrange the ordering of the user reference manual, but for the moment this'll do.

From: **Felix Gaertner**
Date: 15 Mar 1995

Eitan Gurari writes: The example at the end of this note illustrates how the above specifications can be realized in the TeX-based system ProTeX.

1. The `code part' uses the following tagging conventions.

a. External titles for code fragments \<....\>

b. Delimiters for code fragments >>>

c. Internal titles for code fragments `<....`>

2. The `manual part' uses the following tagging conventions.

a. External titles for code fragments \<....\>

b. Delimiters for code fragments ///

c. Internal titles for code fragments @<....@>

3. The `code part' transfers messages to the `manual part' indirectly through an intermediate file.

In the current context, the different tagging conventions enable `code part' to embed code fragments of `manual part' within its code fragments. In more general cases, such conventions allow for hierarchical definitions of literate programs. Families of programs and documents can be defined from a `code part' and a `manual part' by parameterizing them with a common set of tags.

I don't think I quite understand this fully: What is the difference between external and internal titles? Are external titles the tags that are used to reference those text bits from other files? And are internal titles section names like in traditional literate programming systems? If so, why do you make this distinction?

From: **Eitan Gurari**
Date: 17 Mar 1995

Felix Gaertner writes: I don't think I quite understand this fully: What is the difference between external and internal titles? Are external titles the tags that are used to reference those text bits from other files? And are internal titles section names like in traditional literate programming systems? If so, why do you make this distinction?

I tried, apparently unsuccessfully, to use the external and internal terminology to differentiate titles that appear outside code fragments from titles that appear within code fragments. For instance, the above convention implies code fragments of the following form.

```
\<external title associated with the following fragment\>
      ........
      `<internal title referencing code segment 1`>
      ........
      `<internal title referencing code segment 2`>
      ........
                  `%code segment = concatenation of code fragments`%
 >>>
```

ProTeX does not require different delimiters for internal and external titles. On the other hand, a freedom to choose delimiters for titles and code fragments has quite a few applications. Example 1. The ability to represent hierarchies of literate code (my previous posting). Example 2. The ability to request different interpretations to different code fragments within the same literate program.

```
   ........
   \<title for fragment that should be scanned in standard (verbatim) mode\>
      ........
   >>>
   ........
   \[title for fragment that should be scanned through math filter\]
      ........
   >>>
   ........
```

Example 3. The ability to adjust the literate programming notation to the programming language in use. For instance, since `<<' is a frequently used operator in many C++ programs, a notation of the form <<...>> doesn't seem to be natural for titles in such programs.

From: **Marc Mengel**

Date: 28 Mar 1995

I have been thinking about the possibility of 'meta literate programming' to solve this problem: consider the user manual as 'code', which has to be 'documented' together with its implementation. So the description of the user interface is derived from a literate program,just like its implementation! (Presumably, it can be done in TeX? With a WYSIWYG wordprocessor I foresee tremendous problems.)

Colman Reilly writes: Yes. I've done it with TeX on a UNIX system: it's actually just a matter of extracting stuff using sed, so far. I haven't done anything sophisicated yet, so all that happens is that you enclose your user documentation between \begin{user}..\end{user} pairs and it gets extracted and placed in another file to form the user manual.

I have an update to nuweb to support this. I've been planning to release it, this seems as good an excuse as any. I've simply added some new scrap types. nuweb originally had only one scrap type -- verbatim text. I am adding two more, paragraph mode text and math mode text, which are delimited by @[...@] and @(...@) respectively. so in

```
@d scrap1 @{
        some stuff in verbatim mode
@}

@d scrap2 @[
        some stuff in paragraph mode
@]

@d scrap3 @(
        some stuff in math mode
@)
```

The only difference between these is how they are typeset in the main program document. This lets you have typeset document fragments that are extracted in their source form into another document, like a users guide, etc. I am also now considering letting these nest, so that you could have verbatim text with formulas set into it, etc. This way the literate programming tool is still generic and supports arbitrary programming languages, but lets you explicitly prettyprint code, to some extent. Could people let me know if this sounds worthwhile? Something like:

```
@d foo @{
        for ( @(i = 0, i < 100, i++@)) {
                printf("%d\n", @(i@));
        }
@}
```

To typeset the "i = 0, i < 100, i++" in math mode, as well as the "i" in the printf in mathmode, the rest verbatim. This way, the author has control over the typesetting, and the tool still doesn't have to be able read the code. Anyhow the new nuweb version 0.90 is available.

Felix Gaertner writes: Joachim's example seems to be quite common: You want error messages to show up in the user manual. But naturally they are part of the program code. Solution one: You copy the messages by hand into the user manual and format them nicely. But then you're stuck if the code changes. It would be nice to be able to reference these code bits from the user manual, so that the text processor automatically copies them into the right spot when you're looking at it.

Gee, sounds like what you need is a shared include file... The error messages can be included in web source for both the users guide and the code, assuming they are separate documents.

From: **Carsten Tinggard Nielsen**
Date: 30 Mar 1995

Felix Gaertner writes: Joachim's example seems to be quite common: You want error messages to show up in the user manual. But naturally they are part of the program code. Solution one: You copy the messages by hand into the user manual and format them nicely. But then you're stuck if the code changes. It would be nice to be able to reference these code bits from the user manual, so that the text processor automatically copies them into the right spot when you're looking at it.

Marc Mengel writes: Gee, sounds like what you need is a shared include file... The error messages can be included in web source for both the users guide and the code, assuming they are separate documents.

Well there could be an other alternative, and that is macros. In WEB (the original) there was macros due to the lack of some essential

features of Pascal. The macros was dropped in CWEB since they are part of the C preprocessor. Consider the following example of macros (meta?) for both text and code.

```
- - - - - - - - - - -
Define simple non-parameter text macros that can be used both
in text and code scraps.

%define a macro for an error message
@d errormsg Invalid pointer operation

% the usage of the macro in a text scrap
@t
If the given pointer is nil/null/0 then the error message
\textbf{@errormsg} is displayed.

% usage of the error message in the source code
@c code-example
 if (*p == 0)
   error("@errormsg");
- - - - - - - - - - -
```

The above definition assumes the macro text to be "Invalid pointer operation" (text until the end of line). Maybe a more generic for could be:

```
@d errormsg |A longer error
message over more than one line|
```

The macro content is defined inside a pair of ||

From: **Eitan Gurari**
Date: 30 Mar 1995

Carsten Tinggard Nielsen writes: Well there could be an other alternative, and that is macros.

An option already existing (see example) for literate programming systems that are implemented in TeX: the (La)TeX-macros.

In WEB (the original) there was macros due to the lack of some essential features of Pascal. The macros was dropped in CWEB since they are part of the C preprocessor.

Considering that code scraps can by themselves be viewed as macros, in either case we get a multi-types macro system. And conversely, one can also view the additional code macro facilities as new types of code scraps in which one separates their definitions from their listings. Hence, we get a multi-types scrap system. (In earlier posting I tried to illustrate some advantages of such systems.)

```
EXAMPLE
-------
```

```
> Define simple non-parameter text macros that can be used both in text
> and code scraps.
>
> %define a macro for an error message
> @d errormsg Invalid pointer operation
>
> % the usage of the macro in a text scrap
> @t
> If the given pointer is nil/null/0 then the error message
> \textbf{@errormsg} is displayed.
>
> % usage of the error message in the source code
> @c code-example
>  if (*p == 0)
>    error("@errormsg");
```

```
\input ProTex.sty  \def\CodeDel{{}{>>>}}
\AlProTex{,<<<>>>,title,list}

Define simple non-parameter text macros that can be used both in text
and code scraps.

%define a macro for an error message
\def\errormsg{Invalid pointer operation}

% the usage of the macro in a text scrap

If the given pointer is nil/null/0 then the error message
\textbf{\errormsg} is displayed.

% usage of the error message in the source code
\
 if (*p == 0)
   error("\errormsg");
>>>
```

```
> The above definition assumes the macro text to be "Invalid pointer
> operation" (text until the end of line). Maybe a more generic for
> could be:
>
> @d errormsg |A longer error
> message over more than one line|
>
> The macro content is defined inside a pair of ||
```

```
\def\errormsg{A longer error
message over more than one line}
```

Names of 'chunks'

From: **Robert Partington**
Date: 13 Mar 1995

I'm currently writing my third year project in noweb, and I've come across a 'problem'. My project consists of about 8 C source files (and header files) each of which is going to be nowebified. I want the final 'report' to be a TeX file consisting of all the noweb sources included with some other TeX stuff IE.

```
{introduction}
\include life.tex
{more wibble}
\include useri.tex
{conclusion}
```

but I've got a problem with the naming of the chunks, in that most of the source files have an overview which looks like:

```
<<life.n>>=
<<header files>>
<<global variables>>
<<functions>>
@
```

which is going to lead to big name conflicts when all the files get included into one big TeX file. Either that or they will all be cross referenced together, which I don't want. The best solution I can come up with is:

```
<<life.n>>=
<<life:header files>>
<<life:global variables>>
<<life:functions>>
```

@

but I'm not sure that this is a good idea or not. So, to all the people who've written big programs in noweb, I'd like your advice on how to go about it in a sensible, and easy to read/understand (when TeX'd) way.

From: **Norman Ramsey**
Date: 13 Mar 1995

Robert Partington writes: I'm currently writing my third year project in noweb, and I've come across a 'problem'. Most of the source files have an overview which looks like:

```
<<life.n>>=
<<header files>>
<<global variables>>
<<functions>>
@
```

which is going to lead to big name conflicts when all the files get included into one big TeX file. Either that or they will all be cross referenced together, which I don't want.

No. If you are using the LaTeX macros that come with noweb, all you have to do is run noweave over each source file separately. You will see, for example, 8 different instances of <<header files>> in the chunk index, each with a different page number. Identifier cross-reference may be more problematic. Oren Ben-Kiki posted some changes a while back that my provide a little help, but a good, general solution is several months away.

From: **Simon Clift**
Date: 14 Mar 1995

Robert Partington writes: The best solution I can come up with is:

```
<<life.n>>=
<<life:header files>>
<<life:global variables>>
<<life:functions>>
@
```

but I'm not sure that this is a good idea or not.

That's roughly the solution I use. I have a number of boilerplate files for writing my Fortran-90 noweb code. When I write a new function, subroutine, or module, I just pull the boilerplate into the source, and substitute a key word in the boilerplate with the name of the subroutine. This tends to lead to very verbose chunk names, but they tend to be generated automatically, and never actually typed. It also helps me keep chunk names, and other programming practices (e.g. error handling) consistent over projects.

One drawback of this verbosity is that the noweb weaver occasionally duplicates its tags in TeX. My Sparse_Matrix_Tests and Sparse_Matrix_Transforms modules, and Sparse_Order_Spectral_Methods and Sparse_Order_Method_Interface modules generated the same LaTeX labels for the root chunks... (perhaps that's fixed in 2.7 :-}) My boilerplate for an F90 subroutine, using noweb, is given below. I just substitute "FSubName" with my subroutine name and its ready to go. The {parlist} and \bitem{[[stuff is a LaTeX environment I tinkered together for doing parameter listing.

```
\subsection{Subroutine [[FSubName]]}

[[FSubName]] does...

<<Subroutine [[FSubName]]>>=

SUBROUTINE FSubName( input, inout, output, work, err_num )

   <<[[FSubName]] Calling Parameters>>
   <<[[FSubName]] Local Variables>>
   <<[[FSubName]] Code>>

CONTAINS

   <<[[FSubName]] Local Subroutines>>
```

```
END SUBROUTINE FSubName

@ %def FSubName
@

\subsubsection{[[FSubName]] Calling Parameters}

\noindent Input

\begin{parlist}
\end{parlist}

\noindent Input/Output

\begin{parlist}
\end{parlist}

\noindent Output

\begin{parlist}
\end{parlist}

\noindent Work Space

\begin{parlist}
\end{parlist}

\noindent Error Control

\begin{parlist}
\bitem{[[err_num]]} Error code
\end{parlist}

<<[[FSubName]] Calling Parameters>>=

    INTEGER(KIND=4)                 :: err_num

    INTENT(IN)                      ::
    INTENT(INOUT)                   ::
    INTENT(OUT)                     :: err_num

    OPTIONAL                        ::
@

\subsubsection{[[FSubName]] Local Variables}

\noindent Error Control

\begin{parlist}
\bitem{[[sub_name]]}   Subroutine name.
\bitem{[[err_msg]]}    Error reporting supplementary message string.
\bitem{[[err_lbl]]}    Error reporting label for detailed location.
\bitem{[[astat]]}      Allocation status.
\end{parlist}

\noindent Computational Fields/Vectors

\begin{parlist}
\bitem{[[]]}
\end{parlist}

\noindent Loop Control

\begin{parlist}
\bitem{[[i,j]]}       General loop variables
\bitem{[[ineq,inja]]} General matrix loop variables
\end{parlist}
```

```
\noindent Parameters

\begin{parlist}
\bitem{[[]]}
\end{parlist}

<<[[FSubName]] Local Variables>>=

   CHARACTER(LEN=*),PARAMETER               :: sub_name = "FSubName"
   INTEGER(KIND=4)                          :: err_lbl
   CHARACTER(LEN=80)                        :: err_msg
   INTEGER(KIND=4),DIMENSION(:)             :: astat

   INTEGER(KIND=4)                          :: i,j
   INTEGER(KIND=4)                          :: ineq,inja
@

\subsubsection{[[FSubName]] Code}

<<[[FSubName]] Code>>=

    IF ( blah ) GOTO 10 ! *** Error ***
@

Error targets for fatal exception branches.

<<[[FSubName]] Code>>=

10  err_num = 99 ; err_lbl = 10
    err_msg = "Error"
   CALL Error_Print( err_num, mod_name, sub_name, err_msg, err_lbl ) ; RETURN
@
```

From: **Norman Ramsey**
Date: 16 Mar 1995

Simon Clift writes: One drawback of this verbosity is that the noweb weaver occasionally duplicates its tags in TeX. My Sparse_Matrix_Tests and Sparse_Matrix_Transforms modules, and Sparse_Order_Spectral_Methods and Sparse_Order_Method_Interface modules generated the same LaTeX labels for the root chunks... (perhaps that's fixed in 2.7 :-})

Perhaps not :-(Noweb uses the old FORTH trick of taking the length of the name and the first three letters. I'm unlikely to change it. You can tinker with the make_key function in noidx if you want something else...

An extended web language

From: **Carsten Tinggard Nielsen**
Date: 20 Mar 1995

This posting is the first of three describing an extended web language, which I think can solve the problem with doubling the documentation and code in reference and maintenance documents etc.

```
Posting 1: Presentation of the idea
Posting 2: The syntax for extended web
Posting 3: Fragments of an example
```

The basic idea in extended web is that every scrap (text or code) can have at least one type. When weave is run on the LP-src then only those scraps which have the correct type are copied into the weaved files. Eg. every scrap which is compatible with the type Ref is part of the reference document and only those. Here is an example of an extended web-file: The example shows a declaration and implementation of a simple library class in C++.

```
  Produced documents/files:
    reference manual - public interface
    inheritance manual - developers interface
    maintenance manual
    header file
    implementation file
```

The type is marked in a pair of []. The ALL type is compatible with all types (hence the name). The Ref type is for scraps belonging to the reference manual and the Usr type is for the users manual (inheritance guide for subclassing).

```
Scrap Number       Type(s)          Content
-------------------------------------------------
Text 1             [ALL]            Overall description
Text 2             [Ref]            Specific to reference manual
Text 3             [Usr]            Specific to inheritance manual
OutputCode A       [Code]           Declare header file
  <ref to code 1>
OutputCode B       [Code]           Declare implementation file
  <ref to code 5>
Code 1             [Code,Ref,Usr]   Declare the class
  <ref to code 2>
  <ref to code 3>
  <ref to code 4>
Text 4             [ALL]            General description of class
Text 5             [Code]           Private part of class
Code 2             [Code]           Private declarations
Text 6             [Usr,Code]       Protected part of class
Code 3             [Usr,Code]       Protected declarations
Text 7             [Ref,Usr,Code]   Public part of class
Code 4             [Ref,Usr,Code]   Public declarations
Text 8             [Code]           Description of the implementation
Code 5             [Code]           The implementation of class
```

Now what should be in each document? In the reference manual there should be all scraps that is compatible with the type Ref. Any referenced scraps from a code scrap must also be compatible with the type Ref. If not, then the reference is removed.

```
Scraps in the reference manual:
-------------------------------------------------
Text 1             [ALL]            Overall description
Text 2             [Ref]            Specific to reference manual
Code 1             [Code,Ref,Usr]   Declare the class
  <ref to code 4>
Text 4             [ALL]            General description of class
Text 7             [Ref,Usr,Code]   Public part of class
Code 4             [Ref,Usr,Code]   Public declarations
```

Since the reference manual contains only the public interface then the references to Code 2 and Code 3 in Code 1 are removed by the weaver. The types of Code 2 and Code 3 are not compatible with Ref.

```
Scraps in the inheritance manual:
-------------------------------------------------
Text 1             [ALL]            Overall description
Text 3             [Usr]            Specific to inheritance manual
Code 1             [Code,Ref,Usr]   Declare the class
  <ref to code 3>
  <ref to code 4>
Text 4             [ALL]            General description of class
Text 6             [Usr,Code]       Protected part of class
Code 3             [Usr,Code]       Protected declarations
Text 7             [Ref,Usr,Code]   Public part of class
Code 4             [Ref,Usr,Code]   Public declarations
```

The inheritance manual should not have any private parts included; therefore the reference to Code 2 in Code 1 is removed.

```
Scraps in the maintenance manual: (if such is wanted)
-------------------------------------------------
Text 1             [ALL]            Overall description
```

```
OutputCode A       [Code]           Declare header file
  <ref to code 1>
OutputCode B       [Code]           Declare implementation file
  <ref to code 5>
Code 1             [Code,Ref,Usr]   Declare the class
  <ref to code 2>
  <ref to code 3>
  <ref to code 4>
Text 4             [ALL]            General description of class
Text 5             [Code]           Private part of class
Code 2             [Code]           Private declarations
Text 6             [Usr,Code]       Protected part of class
Code 3             [Usr,Code]       Protected declarations
Text 7             [Ref,Usr,Code]   Public part of class
Code 4             [Ref,Usr,Code]   Public declarations
Text 8             [Code]           Description of the implementation
Code 5             [Code]           The implementation of class

Scraps in the header file:
-------------------------------------------------
OutputCode A       [Code]           Declare header file
Code 1             [Code,Ref,Usr]   Declare the class
Code 2             [Code]           Private declarations
Code 3             [Usr,Code]       Protected declarations
Code 4             [Ref,Usr,Code]   Public declarations

Scraps in the implementation file:
-------------------------------------------------
OutputCode B       [Code]           Declare implementation file
Code 5             [Code]           The implementation of class
```

In the next posting I'll present my proposal for the extended web syntax.

From: **Carsten Tinggard Nielsen**
Date: 20 Mar 1995

This is the second posting of three describing an extended web language.

```
Posting 1: Presentation of the idea
Posting 2: The syntax for extended web
Posting 3: Fragments of an example
```

The idea is that both documentation and source are entered in scraps, and each scrap is given one or more types. A view is specified as a filename and one or more types. The resulting view file consists of all the scraps that have the correct type and nothing more.

The @ character denotes some scrap or definition as usual. A scrap is terminated by the next scrap definition. That means that code-scraps are not enclosed in @{ @} like nuweb. @@ is not a command or scrap definition but one @ in the output file.

```
Cmd: @i source-file-name
     @i stddefs.ew
  Source file inclusion

Cmd: @o file-name flags types
     @o foo.cc -t3 [Code]
  Output scrap. Defines the file foo.cc where tabs are converted to
  three spaces. The scrap may have default type Code.
  flags:
    -t# sets the tab width to #
    -i  no indentation of scraps
    may be a lot of other flags.

Cmd: @< reference to code scrap @>
     @< foo private declaration@>
  A reference to a code scrap from a code scrap.

Cmd: @[reference to code scrap@]
     In scrap @[foo declare public methods@] the methods for ...
```

```
  Define a reference to a code scrap from the text.

Cmd: @c code-scrap-name types code-scrap identpart
     @c foo class def [Code,Usr,Ref]
       class foo { ...}
       ...
     @| foo
  Define a code scrap by name and with types
  The type Code may be omitted since the scrap is a default a code scrap.
  The identpart (@|) is optional.

Cmd: @# scrapname types
       where # is 0..9
       @0 part
       @1 chapter
       @2 section
       @3 subsection
       @4 subsubsection
       ...
       @9 paragraph

     @2 Reference for class foo [Ref]
     @2 Developers guide for class foo [Usr]
  Define a text scrap with a 'section' name and type.
  If types are omitted then the scrap has the type [ALL].

Cmd: @v viewname types
     @v fooref.tex [Ref]
     @v foousr.tex [Usr]
     @v foocode.tex [Code]
  Define a file to contain all scraps compatible with the given types.
  The file fooref.tex contains all scraps compatible with type Ref.

Cmd: @t types text
     @t [Usr,Ref] The foo class implements
     the concept of etc etc.
  Definition of a text scrap that has no 'section' name.
  If types are omitted then the scrap has the type [ALL].

Cmd: @<SPACE> types text
     @ [Usr,Ref] The foo class implements
     the concept of etc etc.
  Same as @t.

Miscellanous commands (like nuweb):
Cmd: @f
  Insert all defined output and view files
Cmd: @u
  Insert section with user identifiers, references to etc.
Cmd: @s
  Insert section with all scrap names

Command summary
Command                                 Purpose
-----------------------------------------------------------------
@i source-file-name                     Source file inclusion
@o file-name flags types                Output file scrap
@< reference to code scrap @>           Reference to scrap from code
@[ reference to code scrap @]           Reference to scrap from text
@c code-scrap-name types code-scrap     Define code scrap
     identpart
@N scrapname types (N: 0..9)            Section text scrap
@v viewname types                       Define a file with scraps of types
@t types text                           Text scrap of types
@<SPACE> types text                     Same as @t
@f                                      Section with defined files
@u                                      Section with userdefined identifiers
@s                                      Section with scrap names
```

The syntax can always be discussed (may be a religious matter), so that is not my intention. My intention is to present the idea of scraps with at least one type in order to have multiple 'views' on one literate source file. The syntax for sectioning scraps may be awkward by first sight, but I think it enables you to have multiple resource files defining the actual output of for example a @0 and @4 command. Not everybody works with TeX or LaTeX(2e) so a generic syntax for sectioning ought to have a chance! The reference command from text @[@] may be new. I sure miss it when I'm working with nuweb so here it is. The consequence is that each code scrap should have a label or number.

A word of implementation: I'm not going to implement eweb right-away. The syntax and idea is given to you (as the reader) and my hope is that someone will implement it. Ok, may be I'll do it in a cold winter evening!

From: **Carsten Tinggard Nielsen**
Date: 20 Mar 1995

This is the third and last posting of three describing an extended web language.

```
Posting 1: Presentation of the idea
Posting 2: The syntax for extended web
Posting 3: Fragments of an example
```

The layout of the example follows the outline of the example from the first posting (repeated below). Please notice that if a scrap has no type attached to it, then eweb works exactly like noweb/nuweb. Here is an example of an extended web-file: The example shows a declaration and implementation of a simple library class in C++.

```
    Produced documents/files:
    reference manual - public interface
    inheritance manual - developers interface
    maintenance manual
    header file
    implementation file

Scrap Number       Type(s)           Content
-------------------------------------------------
Text 1             [ALL]             Overall description
Text 2             [Ref]             Specific to reference manual
Text 3             [Usr]             Specific to inheritance manual
OutputCode A       [Code]            Declare header file
  <ref to code 1>
OutputCode B       [Code]            Declare implementation file
  <ref to code 5>
Code 1             [Code,Ref,Usr]    Declare the class
  <ref to code 2>
  <ref to code 3>
  <ref to code 4>
Text 4             [ALL]             General description of class
Text 5             [Code]            Private part of class
Code 2             [Code]            Private declarations
Text 6             [Usr,Code]        Protected part of class
Code 3             [Usr,Code]        Protected declarations
Text 7             [Ref,Usr,Code]    Public part of class
Code 4             [Ref,Usr,Code]    Public declarations
Text 8             [Code]            Description of the implementation
Code 5             [Code]            The implementation of class

The example
-------------------------------------------------------------------
% define the views:
@v fooref.tex        [Ref]
@v foousr.tex        [Usr]
% all files uses standard definitions

% Text 1             [ALL]           Overall description
@t \input{stddef.tex}

% Text 2             [Ref]           Specific to reference manual
@2 Reference manual for class foo [Ref]
```

```
% Text 3             [Usr]           Specific to inheritance manual
@2 Guide for subclasses of class foo [Usr]

% OutputCode A       [Code]            Declare header file
%  <ref to code 1>
@o foo.h -i
#ifndef FOO_H
#define FOO_H
  @<foo class declaration@>
#endif FOO_H

% OutputCode B       [Code]            Declare implementation file
%  <ref to code 5>
@o foo.cc -i -t3
#include "foo.h"
  @<foo class implementation@>

% Code 1             [Code,Ref,Usr]    Declare the class
%  <ref to code 2>
%  <ref to code 3>
%  <ref to code 4>
@c foo class declaration
class foo {
  @<foo declare private part@>
  @<foo declare protected part@>
  @<foo declare public part@>
};
@| foo

% Text 4             [ALL]             General description of class
@ The foo class implements ...

% Text 5             [Code]            Private part of class
@ [Code]
The two instance variables are declared ...

% Code 2             [Code]            Private declarations
@c foo declare private part
private:
  int flip;  // direction: <0 is back
  char curchar; // current character from input
  void initialize(int f, char c);

% Text 6             [Usr,Code]        Protected part of class
@ [Usr,Code]
The update methods can be used by friends of this class

% Code 3             [Usr,Code]        Protected declarations
@c foo declare protected part [Usr,Code]
protected:
  void updateFlip(int newFlip);
  void setNextChar(char newChar);

% Text 7             [Ref,Usr,Code]    Public part of class
@t [Ref,Usr,Code]
The foo class can be created by either default or explicit
initialization ...

% Code 4             [Ref,Usr,Code]    Public declarations
@c foo declare public part [Ref,Usr,Code]
public:
  foo();
  foo(int Iflip = 0, char Icurchar = ' ');
  ~foo();

% Text 8             [Code]            Description of the implementation
@t [Code]
The implementation of the foo class ...
```

```
% Code 5             [Code]            The implementation of class
@c foo class implementation
void foo::initialize(int f, char c) {
  flip = f; curchar = c;
}

// constructors
foo::foo() {
  initialize(0, ' ');
}
foo::foo(int Iflip, char Icurchar) {
  initialize(Iflip, Icurchar);
}

// destructor
foo::~foo() {}

// updating methods
void foo::updateFlip(int newFlip) {
  flip = newFlip;
}

void foo::setNextChar(char newChar) {
  curchar = newChar;
}

================================================
The content of the reference manual (fooref.tex):
  (this is showed by using an output that could be sent to nuweb)

% Text 1             [ALL]           Overall description
\input{stddef.tex}

% Text 2             [Ref]           Specific to reference manual
\section{Reference manual for class foo}

% Code 1             [Code,Ref,Usr]    Declare the class
%  <ref to code 4>
@d foo class declaration @{
class foo {
  @<foo declare public part@>
};
@| foo@}

% Text 4             [ALL]             General description of class
The foo class implements ...

% Text 7             [Ref,Usr,Code]    Public part of class
The foo class can be created by either default or explicit
initialization ...

% Code 4             [Ref,Usr,Code]    Public declarations
@d foo declare public part @{
public:
  foo();
  foo(int Iflip = 0, char Icurchar = ' ');
  ~foo();
@}
```

From: **Sven Utcke**
Date: 22 Mar 1995

This certainly sounds like a good idea to me. Although I would rather like to see it imbedded in FWEB and think the notation slightly awkward, it is otherwise precisely what I've been waiting for all along (together with html support)! What do other people think?

From: **Joachim Schrod**
Date: 23 Mar 1995

Your article is very good. Nevertheless I would like to explicate the problem domain of your solution. As the `clients' of your code are programmers, you don't need a different `language' for presentation. In addition, you just produce the documents for one module. The problem domain of user manuals for a medium-sized application (let's say, in the range of 2-20 MB source) where the (graphical) user interface is realized by thousands of widgets in several modules *cannot* be attacked in the way you outlined it.

Furthermore I still don't know how I shall code several implementations for one module, looking at your proposal. (I.e., several implementations of one specification.) Shall I repeat the specification in all these documents? It would destroy maintainability. Shall I store all implementations in one document? I would yield sources with sizes in the MB range, and my SCM would be near to useless. Sorry, while I'm very sure that your proposals are very good for our student projects, I'm afraid they don't help us at all in our real developments.

From: **Norman Ramsey**
Date: 25 Mar 1995

John Bentley and some others at Bell Labs tried something like this for a language definition. They got something like 4 or 5 different presentations out of it. Unfortunately, this tack seems to work only when all the documents we're talking about have the same structure. Even then it is awkward without good editing tools. I hope we will see more research on maintaining multiple consistent documents and programs from a single source, but I think it is a hard problem and that we will need special editing support.

From: **Dieter Baron**
Date: 27 Mar 1995

Somehow this whole problem (at least if the documents are of the same structure) reminds me of writing code that is meant to compile on different platforms. The approach of your language is analogous to #ifdefs that say which parts to include on what platform (in what view). With a larger number of platforms and larger documents, this approach becomes unwieldy -- at least in my expierience. But maybe a folding editor would help, displaying only the parts relevant to the current view. Anyone to write an Emacs mode?

Maybe a tool to extract interface definitions -- like prototypes in C -- to include them in the proper place of the user manual -- kept in a separate file -- might be a better solution for larger projects. It would making reordering the functions to fit the flow of the explanation easier, and long conceptual introductions would not clutter up the code file. However, one would probably want this tool to mention functions that have changed since the last run, so one knows what sections of the user manual to update.

From: **Marc Mengel**
Date: 28 Mar 1995

I was seriously considering an idea very much like this, and I have now punted on it. The reason is that a single fragment *may* be usable in multiple contexts which would entail different types. For example, the same fragment may want to be in the users guide, design document, and in a help message in the code. In a system like nuweb you can do something like a definition of a scrap "help message"

```
@d help message
@[To use the frobnitz deframulator, press the "defram" button on
the main screen. @]

could be used in a perl script:

@o foo.pl @{
        ...
        print <<EOF
@<help message @>
EOF
        ...
@}

and in a document

@o usersguide.tex @{
        ...
        @<help message@>
        ...
@}
```

From: **Carsten Tinggard Nielsen**
Date: 29 Mar 1995

There have been some comments on my three articles, and I appreciate the comments and objections, and consider them very usefull. Joachim Schrod does point out several weak points about what to do with large scale (multi platform) projects, what to do about files etc. when using web or an extended form of web. As pointed out by Kasper Osterbye, there might be a far better way using a hypertext tool that can be configured as you like with different views and output configurations.

I don't consider web to be the silver bullet, but a tool in the right direction. However I saw a problem when writing library classes and the demand for at least two different manuals or views on a library class. And I agree that this is not large-scale programming at all; there should be room for a lot of other tools as well.

From: **Felix Gaertner**
Date: 29 Mar 1995

The real problem of writing several documents at the same time and keeping them consistent is the fundamental difference in the structure of, say, a reference manual, a tutorial and module interface description. If you try to squeeze all these types of documents into a single source file, you will (unwillingly) try to assimilate the structure of these documents and make them unnatural.

It seems to me that the traditional WEB-way of document writing (documentation and code) only works, because you are really only writing one document! Don't forget that traditionally TANGLE produces stuff for compilers and not for humans. Writing several documents in this way is bound to fail. (There are more reasons that others have already mentioned.)

But the problem of doubling documentation still needs to be tackled. The central `axiom' should still be ``define things only in one place'' and the solution has to be more than merely using `shared include files'. Maybe we can think of the documentation as documents that share common paragraphs or sentences. I still like the notion very much of representing this as a hypertext, where the referenced text bits are shown inline. (Such a solution was proposed earlier in this newsgroup by Kasper Osterbye.) To get a printed document you only need to know in which place to start and and algorithm to linearize the text references. [BTW: Is this only an editor problem?]

From: **Stephen Boyan**
Date: 30 Mar 1995

I would like to repeat an earlier comment relating to this thread about multiple documents from one source file. I do not think a web is designed for this if n > 2, where n is the number of outputs. I think literate programming is about writing a program like a textbook, then processing the source to create both a textbook and a program. I think this is the main design goal - an executable for a machine, and an explanation of the design of the executable for a human, both created from one source. If one gets too far away from this, a web will not be an appropriate tool. If n is close to 2, then maybe a web will work; but it's not what it was designed for.

There is a class of tools that is designed for managing products that comprise multiple outputs from multiple inputs. Each of the various documents and executables in a set relating to a program or system product can be considered a configuration, and a configuration manager seems to me a better choice of tool for handling this. Perhaps many of the configurations could then be processed through a web to give a document and an executable. This can also be controlled by the configuration manager. A make processor is a very simple form of this kind of tool, but there are more robust - and expensive - examples. One can then tie together program source, manuals, test data for regression testing, and other artifacts.

From: **Adelbert Fernandez**
Date: 23 May 1995

Greetings programmers, I'm been studying up on literate programming, getting ready to take the plunge. There's one thing I'm expecting out of literate programming that I'm not seeing in the examples. Literate programming seems to focus (at least the examples I've seen so far) on documenting the implementation of a program. I tend to think and work in components (e.g. classes and libraries), so to me documenting the component interface is at least as important as documenting the implementation.

By using literate programming, I'm expecting to head off two problems:

1) the interface document stays in sync with the interface

2) the interface (.h) stays in sync with implementation (.c)

#2 is not normally a problem if one uses prototyping and good typing practices, however, my project is firmly embedded in the use of non-prototyped K&R C, and yet strangely, uses a simulated OO approach that relies on the use of many function ptrs to confuse even lint.

Ideally, I'd expect my literate propgramming process to be something like this (for component xxx):

1) Write xxx.web file.

2) Tangle xxx.web to produce xxx.h, xxxP.h, and xxx.c.

3) Weave xxx.web to produce xxx-interface.tex and xxx-implementation.tex.

4) Hand xxx-interface.tex, xxx.h, and libxxx.a to client user.

5) Hand xxx-implementation.tex, xxx.h, xxxP.h, xxx.c, and xxx.web to new component maintainer.

Is the mapping of .web files to output files normally 1-to-1, e.g. xxx.web is only meant to produce xxx.c, but not both xxx.h and xxx.c? Is 1-to-N mappings a problem? Does this process fit the literate programming paradigm? Is this anywhere close to how some of you experienced literate gurus work? Any comments are welcome.

From: **Lee Wittenberg**
Date: 24 May 1995

Bert Fernandez writes: Is the mapping of .web files to output files normally 1-to-1, e.g. xxx.web is only meant to produce xxx.c, but not both xxx.h and xxx.c? Is 1-to-N mappings a problem?

1-to-N mappings are common in both CWEB and noweb, to my certain knowledge, and I believe that most of the other tools support it as well. In CWEB, the @(code does the work, an your xxx web (in the filexxx.w) would look something like this (text chunks removed):

```
@c
        /* this code is extracted to file xxx.c */

@(xxx.h@>=
        /* this code is extracted to file xxx.h */

@(xxxP.h>=
        /* this code is extracted to file xxxP.h */
```

The multiple weaving is more difficult, but possible with judicious uses of \if's. I'm not sure if this is really what literate programming is for, though. Literate programming is really designed for the component programmers, not component users. Norman Ramsey has a neat noweb trick of including the troff code for man pages inside \iffalse ... \fi pairs, and tangling the troff code separately (xxx-interace.3, in your example). This could also work in CWEB. noweb is much simpler. You simple name chunks xxx.c, xxx.h, etc. and use notangle's -R option to extract the piece you want.

Does this process fit the literate programming paradigm? Is this anywhere close to how some of you experienced literate gurus work?

Yes. This is *exactly* what I do with my programs, except for the multiple weaves.

From: **Norman Ramsey**
Date: 24 May 1995

Ideally, I'd expect my literate propgramming process to be something like this (for component xxx):

1) Write xxx.web file.

2) Tangle xxx.web to produce xxx.h, xxxP.h, and xxx.c.

3) Weave xxx.web to produce xxx-interface.tex and xxx-implementation.tex.

4) Hand xxx-interface.tex, xxx.h, and libxxx.a to client user.

5) Hand xxx-implementation.tex, xxx.h, xxxP.h, xxx.c, and xxx.web to new component maintainer.

Is the mapping of .web files to output files normally 1-to-1, e.g. xxx.web is only meant to produce xxx.c, but not both xxx.h and xxx.c? Is 1-to-N mappings a problem?

nuweb, noweb, SpiderWeb, Levy-Knuth CWEB, and probably others implement one-to-many tangle transformations. For weaving, I have typically approached (3) in one of two ways:

```
  - make xxx-interface.tex a special kind of ``code chunk''
  - use conditional compilation in tex, so
        noweave xxx.web > xxx.tex
        latex '\let\interface=1 \input xxx'
        latex '\let\implementation=1 \input xxx'
    with suitably gross TeX hacking to make the right thing happen
```

Neither approach makes me happy.

Bizarre and cryptic language

From: **Will Ware**
Date: 29 Mar 1995

I have only recently become aware of literate programming, and am interested in trying it to expedite a large body of software that I need to write at work. I program in C, so I would be using CWEB, but I don't think we have any TeX-compatible printers where I work. Is there such a thing as a TeX-to-Postscript converter? Also, in the library I have found writings on literate programming up to about 1984 or so, and very little after that. Has the rise of personal computers caused literate programming to fall into disfavor? From the distribution of folks posting this newsgroup, it looks like it. Last question, net resources: Is there a literate programming FAQ? Are there libraries of successful and illustrative literate programs? Are there any glaring huge success stories, like "all fifty quadrillion lines of code for the Space Shuttle were done in literate programming"?

From: **Andrew Mauer**
Date: 30 Mar 1995

Will Ware writes: I have only recently become aware of literate programming, and am interested in trying it to expedite a large body of software I need to write at work. I program in C, so I would be using CWEB,

There is no need to use CWEB if you write in C. I would recommend noweb, since it is much easier to learn and I also like the appearance of the source more (less markup junk). It does not pretty print, so that may be a minus or may be a plus.

I don't think we have any TeX-compatible printers where I work. Is there such a thing as a TeX-to-Postscript converter?

TeX outputs dvi files. "dvips" translates them to postscript. This is the usual method.

Last question, net resources: Is there a literate programming FAQ?

ftp://rtfm.mit.edu/pub/usenet-by-group/comp.programming.literate/*

Are there libraries of successful and illustrative literate programs? Are there any glaring huge success stories, like "all fifty quadrillion lines of code for the Space Shuttle were done in literate programming"?

(Would that be a success story? BOOOM!) TeX, Metafont, and CWEB are all huge literate programs written by Donald Knuth. They are impressive but not particularly a good way to learn literate programming. Bell Lab's "lcc" compiler is apparently written in noweb, as is the book about it. The FAQ should have pointers to web sites with examples.

From: **Will Ware**
Date: 30 Mar 1995

I've made a little progress in the past couple days. I found the FAQ, and I've had a chance to look over some documents about both CWEB and CLiP. If I understand what I'm reading, the programmer is expected to work in this bizarre and cryptic language with lots of '@' signs and other odd and meaningless punctuation.

Wouldn't it be easier to do one's literate programming using a wysiwyg word processor (e.g. Word for Windows) and indicate what is source code by putting it in a different font? I have often seen this sort of convention used in textbooks and academic papers. A lot of the work, I think, could be done by a filter program that accepts a Word document and outputs an ASCII file including only the text that was in that font. Does anybody know if the file format for Word documents is available? or is that Microsoft-proprietary?

One thing I haven't figured out is the precise role of macros in things like WEB and CLiP. I have no trouble understanding C preprocessor statements, but I'm not sure what WEB macros do, or are supposed to be capable of doing. Knuth's and others' writings seem to imply that it should be obvious by example. So, OK, color me learning-disabled. But it would seem that, without some kind of interesting macro facility, "literate programming" would just be a tricky name for unusually verbose commenting. I assume it's more than that, so any enlightenment in this vein would be appreciated.

From: **Marc van Leeuwen**
Date: 30 Mar 1995

Will Ware writes: If I understand what I'm reading, the programmer is expected to work in this bizarre and cryptic language with lots of '@' signs and other odd and meaningless punctuation.

Not to mention the C language itself, which has an orgy of bizarre and cryptic operators, delimiters and punctuators, doubtlessly odd and meaningless to the uninitiated. But seriously, the pros and cons of structured markup as opposed to WYSIWYG production of documents is an issue that is unrelated to literate programming. I believe there exists WYSIWYG systems for literate programming, but the processing of marked-up source text seems to be more popular, probably also because it is simpler to implement and more flexible.

Wouldn't it be easier to do one's literate programming using a wysiwyg word processor (e.g. Word for Windows) and indicate what is source code by putting it in a different font? I have often seen this sort of convention used in textbooks and academic papers. A lot of the work, I think, could be done by a filter program that accepts a Word document and outputs an ASCII file including only the text that was in that font. Does anybody know if the file format for Word documents is available? or is that Microsoft-proprietary?

Apart from being highly system dependent, such an approach would be rather error prone. Besides, there really is more to literate programming than just tangling, and there is more to tangling than just selecting the program code (that's why it was called tangling, right?).

One thing I haven't figured out is the precise role of macros in things like WEB and CLiP. I have no trouble understanding C preprocessor statements, but I'm not sure what WEB macros do, or are supposed to be capable of doing. Knuth's and others' writings seem to imply that it should be obvious by example. So, OK, color me learning-disabled. But it would seem that, without some kind of interesting macro facility, "literate programming" would just be a tricky name for unusually verbose commenting. I assume it's more than that, so any enlightenment in this vein would be appreciated.

For an elementary introduction to the concept of literate programming and why one would want to do it, see the first sections of the CWEBx manual, "http://www.tex.ac.uk/tex-archive/web/c_cpp/cwebx/cwebx/manual.tex" For more background, discussed by its inventor, see Knuth book ``Literate programming'', CSLI lecture notes 27, 1992 (ISBN 0-937073-81-4).

From: **Mark Kramer**
Date: 31 Mar 1995

Will Ware writes: I've made a little progress in the past couple days. I found the FAQ, and I've had a chance to look over some documents about both CWEB and CLiP. If I understand what I'm reading, the programmer is expected to work in this bizarre and cryptic language with lots of '@' signs and other odd and meaningless punctuation.

This may be true for CWEB, but certainly not for CLiP. One of the main ideas behind CLiP is that the code chunks should look as natural as possible. So there is no "odd and meaningless punctuation". (Or are the, minimal, CLiP-options considered odd? Meaningless options are never needed!) Furthermore, CLiP was designed to work with any reasonable text processor, especially wysiwyg word processors. That is another reason to avoid explicit commands in CLiP. CLiP does not use a separate LP-language, let alone a "bizarre and cryptic language with lots of '@' signs". I fear CLiP is being blamed for features of another literate programming tool :-(

Wouldn't it be easier to do one's literate programming using a wysiwyg word processor (e.g. Word for Windows) and indicate what is source code by putting it in a different font? I have often seen this sort of convention used in textbooks and academic papers. A lot of the work, I think, could be done by a filter program that accepts a Word document and outputs an ASCII file including only the text that was in that font.

And how should the tool know the order of code chunks? In CLiP, this is exactly the purpose of the special CLiP-lines and the options therein.

One thing I haven't figured out is the precise role of macros in things like WEB and CLiP.

As far as I know CWEB's macro's are just passed to the C preprocessor. In CLiP there are no macro's, but one could include macro's in the code, of course.

From: **Norman Ramsey**
Date: 31 Mar 1995

Will Ware writes: If I understand what I'm reading, the programmer is expected to work in this bizarre and cryptic language with lots of '@' signs and other odd and meaningless punctuation.

One man's bizarre is another man's treasure :-) You might want to check out noweb, which (among other things) was designed to require minimal punctuation. CWEB is definitely an offender in this area. If, however, you get into this business, you will probably latch on to a favorite punctuation which you can enjoy defending rabidly against all comers. :-)

it would seem that, without some kind of interesting macro facility, "literate programming" would just be a tricky name for unusually verbose commenting. I assume it's more than that, so any enlightenment in this vein would be appreciated.

I see it's time for the ``how is literate programming different from verbose commenting'' question. Perhaps David Thompson will get this into the FAQ. Alert! What follows are my opinions. In no way do I claim to speak for the (fractious) literate-programming community. How is literate programming different from verbose commenting? There are three distinguishing characteristics. In order of importance, they are: flexible order of elaboration, automatic support for browsing, and typeset documentation, especially diagrams and mathematics.

Flexible order of elaboration means being able to divide your source program into chunks and write the chunks in any order, independent of the order required by the compiler. In principle, you can choose the order best suited to explaining what you are doing. More subtly, this discipline encourages the author of a literate program to take the time to consider each fragment of the program in its proper sphere, e.g., not to rush past the error checking to get to the ``good parts.'' In its time and season, each part of the program is a good part. (This is the party line; your mileage may vary.)

I find the reordering most useful for encapsulating tasks like input validation, error checking, and printing output fit for humans --- all tasks that tend to obscure ``real work'' when left inline. Reordering is less important when using languages like Modula-3, which has exceptions and permits declarations in any order, than when using languages like C, which has no exceptions and requires declaration before use.

Automatic support for browsing means getting a table of contents,index, and cross-reference of your program. Cross-reference might be printed, so that you could consult an index to look up the definition of an identifier `foo'. With good tools, you might get a printed mini-index on every page if you wanted. Or if you can use a hypertext technology, cross-reference might be as simple as clicking on an identifier to reach its definition.

Indexing is typically done automatically or `semi-automatically', the latter meaning that identifier definitions are marked by hand. Diligently done semi-automatic indexes seem to be best, because the author can mark only the identifiers he or she considers important, but automatic indexing can be almost as good and requires no work. Some tools allow a mix of the two strategies. Some people have applied literate-programming tools to large batches of legacy code just to get the table of contents, index, and cross-reference.

I don't use diagrams and mathematics very often, but I wouldn't want to have to live without them. I have worked on one or two projects where the ability to use mathematical formulae to document the program was indispensible. I also wouldn't like to explain some of my concurrent programs without diagrams. Actually I write almost all of my literate programs using only sections headers, lists, and the occasional table.

Wouldn't it be easier to do one's literate programming using a wysiwyg word processor (e.g. Word for Windows) and indicate what is source code by putting it in a different font?

The data formats used in wysiwyg products are proprietary, and they tend to be documented badly if at all. They are subject to change at the whim of the manufacturer. (I'll go out on a limb and say there are no significant wysiwyg tools in the public domain. I hope the Andrew people will forgive me.) These conditions make it nearly impossible to write tools, especially tools that provide automatic indexing and cross-reference support. The CLiP people have a partial solution that works for tools that can export text --- they plant tags and delimiters throughout the document that enable the reordering transformation (``tangling'').

People use TeX, roff, and HTML because free implementations of these tools are widely available on a variety of platforms. TeX and HTML are well documented, and TeX and roff are stable. TeX is the most portable. I think I have just answered the FAQ ``how come all these tools use TeX, anyway?'' :-)

WEB++

From: **Michal Gomulinski**
Date: 18 Apr 1995

Could anyone tell me how to write big C++ literate programs, using CWEB. I mean how to deal with programs in multiple files and a lot of ".h" files as it usually is done in C++ programming. The natural way in writing CWEB files is to put everything into one file, but I don't think it is the best solution. Thanks for your answers!

From: **Andreas Scherer**
Date: 21 Apr 1995

Writing a single CWEB document and utilizing the `@(filename@>' feature of CTANGLE is one way to deal with large software systems consisting of multiple source files and associated header files. I personally like this approach and both CTANGLE and CWEAVE are large enough even in their standard flavour.

The other way is to write a CWEB document for each of the C(++) modules. An extensive example for this approach is the C(--) implementation of `The Stanford GraphBase' by Donald Knuth. He wrote more than 30 CWEB modules for the associated C modules. He used the `multiple output' mechanism for header files in case that external declarations had to be exported from a C module. The documentation for this collection of literate programs comes as a book of the name `The Stanford GraphBase', where each software module forms a major part or chapter of the printed documentation. The `local' indexes created by CWEAVE were collected in a single index for the whole book, so inter-module references are possible.

In connection with a well-written `Makefile' the second approach can save a lot of time in the development phase, because neither the whole CWEB source has to be processed by CTANGLE (or CWEAVE) nor have all the C modules created automatically to be recompiled after each modification.

From: **Kiyoshi Akima**
Date: 17 Jun 1995

I'm using C++ and CWEB, but this question should be general enough. In illiterate programming a common organization for a library is a single header file giving the interface to the library and several source files implementing the various functions in that library. One of the reasons for the several source files is that if all of the code was in one file, most conventional linkers will pull in all of it even when you only needed a portion of it. For example an application might only need the output functionality of the library without the input functionality.

The library I'm working on is small enough that I would like to document the entire thing in one CWEB file. I know I can use the "@(" control sequence to output to different files, but what should the main file be? I'm thinking of having the unnamed code section go to the header file and using @(to produce the several implementation files. Is this the right way? Does anybody have experience doing this type of thing?

From: **Lee Wittenberg**
Date: 20 Jun 1995

Kiyoshi Akima writes: The library I'm working on is small enough that I would like to document the entire thing in one CWEB file. I know I can use the "@(" control sequence to output to different files, but what should the main file be? I'm thinking of having the unnamed code section go to the header file and using @(to produce the several implementation files. Is this the right way? Does anybody have experience doing this type of thing?

It's too early in the lifetime of literate programming for any Right Way to have surfaces. We all argue about techniques, but I think we all realize that we're all still working in the dark. On the other hand, CWEB is set up to generate a .c file as the main file by default. so your technique is certainly something Knuth didn't really consider (of course, that in itself may be a Good Thing).

Does anybody have experience doing this type of thing?

I sometimes use @(to mimic noweb's multiple-root-chunk facility, and don't even bother to have an unnamed chunk at all. My personal preference, when confronted with multiple outputs (more than just a .h and .c with the same name) is to name all the chunks to be extracted with their file names. In CWEB, this would mean no @c chunks, in noweb, no <<*>> chunk. I think CWEB produces a zero-length .c file, with perhaps, a warning message, but that's only a minor irritant.

From: **Michal Gomulinski**
Date: 21 Jun 1995

Kiyoshi Akima writes: The library I'm working on is small enough that I would like to document the entire thing in one CWEB file. I know I can use the "@(" control sequence to output to different files, but what should the main file be? I'm thinking of having the unnamed code section go to the header file and using @(to produce the several implementation files. Is this the right way? Does anybody have experience doing this type of thing?

Not long ago I finished writing a program in C++ in which I used FWEB. I developed competly different strategy of spliting code into header and code files. I used two types of web files: *.hw and *.w. The first one was literate header file and it was tangled into *.h file by ftangle and *.w was literate code file and was tangled into *.cc. The makefile I used was constructed to recompile parts of the program only if the real source (*.w, *.hw) was changed.

To get woven output of my work I wrote another file, let's say "documentation.w" which contained the introductory part of the documentation, general information about program and several @i"..." statements which included all files, which actually contained the code. After this there was also a "REFERENCES" part. To get the documention I had something like this in the makefile:

```
fweave documentation.w
latex2e documentation
latex2e documentation
latex2e documentation
xdvi documention
```

I think that such oragnisation is quite good for C++ programs and libraries.

C++ templates and FWEB

From: **Come Raczy**
Date: 30 Apr 1995

Come Raczy writes: The problem was to get a nice output, using FWEAVE on C++ generic classes. FWEAVE doesn't seem to understand the notation : MyClass<T>.

Here are the answers, and a partial solution for the problem. I just got two kind of answers: 1) "Don't use C++! Even C++ compilers have problems to understand this messy syntax." OK, a lot of C++ features may be obtained with C, if you use efficiently the preprocessor. Moreover, with FWEAVE you can overload identifiers and operators, so C and FWEAVE can be more efficient than C++. 2) "Don't use FWEAVE! Switch to a non-pretty-printing WEB." OK, emacs can't help you to create "nice" code, then turn on FWEAVE's language independant mode (Cheer up! Flee!:-)).

So I'tried to find the minimal tricks to get what I need. The solution that I hold back uses FWEB's internal macros. For each generic class named "MyClass" I define a macro:

```
@M @-GenericMyClass MyClass
```

It seems to work fine with FWEB 1.40. here is an example: Note: I use @/ at the end of each line "template <class T>" instead of @; because there is less space between the lines, but it wouls work with @; too.

```
@*Main.
@f @-cout int
@a
#include <iostream.h>
@<Declaration of |MyClass|@>@;
@<Implementation of |MyClass|@>@;

int main(void){
MyClass<int> cc(1,2,3);
cout << cc << "\n";
return 0;
}

@*Description of $MyClass$.
```

```
@*1Definition.

@f MyClass int
@f GenericMyClass MyClass
@M @-GenericMyClass @[MyClass<@-T>

@<Declaration of |MyClass|@>=

template <class T>@/
class MyClass{
public:
    MyClass(const GenericMyClass &x);
    friend ostream &operator <<(
        ostream &os, const GenericMyClass &x);
private:
    T v[3];
};

@*1Implementation.

@<Implementation of |MyClass|@>=

template <class T>@/
GenericMyClass::MyClass(const GenericMyClass &x){
for(int i=0; i<3; i++) v[i]=x(i);
}

template <class T>@/
ostream &operator <<(ostream &os, const GenericMyClass &x){
return(cout << "[" << x.v[0] <<", "<<x.v[1]<<", "<<x.v[2]<<"]");}

@*INDEX.
```

From: **David Kastrup**
Date: 02 May 1995

Come Raczy writes: Here are the answers, and a partial solution for the problem. I just got two kind of answers: 1) "Don't use C++! Even C++ compilers have problems to understand this messy syntax." OK, a lot of C++ features may be obtained with C, if you use efficiently the preprocessor. Moreover, with FWEAVE you can overload identifiers and operators, so C and FWEAVE can be more efficient than C++. 2) "Don't use FWEAVE! Switch to a non-pretty-printing WEB." OK, emacs can't help you to create "nice" code, then turn on FWEAVE's language independant mode (Cheer up! Flee!:-)).

So I'tried to find the minimal tricks to get what I need. The solution that I hold back uses FWEB's internal macros. For each generic class named "MyClass" I define a macro:

In fact, I gave *both* kind of answers (namely that using a "simple" syntax-parsing tool on scrap-base will not go well with a messy-syntax with heaps of exceptions and context sensitivity language as C++). The "solution" supports my point. It cannot be the purpose of literate programming, in my opinion, to change the look or feel of a programming language. When you have to "design" your document with the limits of FWEAVE or whatever tool in mind, you cannot use that features of the language which appeal most naturally to the problem. Now this was not meant to say that you should use FWEAVE with C macro brewery instead, but rather use it with some other OO language with less problematic syntax (or rather, a syntax-aware other system with another OO-language).

Or give up the syntax-analysis and concentrate on just explaining your code by using a syntax-independent system. But when you have to fight to make a tool work with your language of choice, I think your time could be better spent. I use noweb for this reason, because it *saves* me time, not because I consider the additional invested time worth it.

Is literate programming useful for object-oriented programming?

From: **Jacob Nielsen**

Date: 14 May 1995

In a desperate attempt to outnumber the off-topic posts in comp.programming.literate, I want to discuss if the literate programming paradigm is appropriate for object-oriented languages. The literate programming paradigm excels in the documentation of large complex systems, but do we need literate programming when using OOP? There's no doubt that documentation is a good thing but is the overhead of scraps, etc. needed to document a system consisting of the small components promoted by OOP?

In my view, documentation for an object-oriented system should consist of: 1. Description of the relationships between the classes/class clusters. Here a suitable modeling technique should be used; e.g. Rumbaugh and/or Booch. 2. How to use the class and each of its methods. 3. A description of the algorithm in each method, but normally there will be no need to split the method into more than one scrap. We can meet these requirements without using the literate programming paradigm.

I have used literate programming in a C++ project and really enjoyed the ability to introduce instance variables when they were first used in a method, but the need and reason for these variables should have been discovered and documented in the analysis phase. My need for scraps were, more or less, limited to containing a method in full. In hindsight, the project could have been documented equally well using good commenting practices and suitable modeling techniques only. I doubt that I will use a literate programming tool in future projects involving OO-languages but I will most certainly document my work to a much larger extent than before I learned of literate programming and realized just how much was missing from my previous documentations.

From: **Marc van Leeuwen**
Date: 15 May 1995

Jacob Nielsen writes: The literate programming paradigm excels in the documentation of large complex systems, but do we need literate programming when using OOP? There's no doubt that documentation is a good thing but is the overhead of scraps, etc. needed to document a system consisting of the small components promoted by OOP?

Just a few comments. Literate programming is not identical with the use of modules (scraps, chunks, ...). In fact these are just a means towards the goal of providing the highest possible level of documentation to the code. They are however very effective in the situation they are intended for, because they achieve two things at once: 1) move code dealing with details that are not discussed in the commentary of the current section outside of its view, so that the connection between the commentary and the (visible) code is more easily perceived; 2) provide a short abstract of the omitted code through its module name. The second point should not be overlooked; sometimes it is so effective that some sections need no commentary, because the module name already says it all (just like some mail messages need no body after stating their subject line). It is also a point that will be missed if one just uses function calls instead of module abstractions as a means of partitioning the code into small pieces; it is not likely that one will use function names with some 100 characters or so, even if the language allows it and the task of the function cannot be described well with fewer characters.

It seems to me that the distinguishing feature of the object-oriented paradigm might well lie not in the methodology itself, but rather in the class of problems that it can effectively be applied to. From my experience it seems that there are some problem areas, in particular event driven programming such as occurs in programs with an elaborate graphical user interface, where one is led almost automatically into an object-oriented approach. Here most effort is directed towards designing a consistent set of data types, and maintaining the necessary invariants of the data, and a proper correspondence with the visual representation presented to the user; this leads to a plethora of small functions and a natural connection between data types and their methods. However, I have never seen an example of a task of any algorithmic complexity (in the every-day sense of the word complexity) that could be handled effectively by object-oriented techniques; I have really no idea how one could apply these techniques to make programs such as TeX more transparent, or (for a less monolithic example) programs like those in the Stanford GraphBase.

My own experience, which ranges form writing mathematical software (computer algebra) and parts of a compiler to some work on a program with an extensive user interface, suggests that the utility of literate programming techniques increases strongly with algorithmic complexity: for some small functions the code is so obvious that there is nothing to make a sensible comment about, while at the other extreme I have had cases where a full A4 page of commentary was attached to a single assignment statement (it dealt mainly with explaining why nothing more than that was needed to handle all conceivable cases). But regardless of the algorithmic complexity, every large project will need an introduction in pure commentary that explains the task that is being performed, and the global structure that underlies the design; such a description is much more readable on typeset pages than as a huge block comment in the code.

In my view, documentation for an object-oriented system should consist of: 1. Description of the relationships between the classes/class clusters. Here a suitable modeling technique should be used; e.g. Rumbaugh and/or Booch. 2. How to use the class and each of its methods. 3. A description of the algorithm in each method, but normally there will be no need to split the method into more than one scrap.

I don't understand what you mean by a modeling technique in point 1. From point 3 it seems to me however that the software you are considering has a very limited algorithmic complexity.

From: **Mark Chu-Carroll**
Date: 16 May 1995

Jacob Nielsen writes: In a desperate attempt to outnumber the off-topic posts in comp.programming.literate, I want to discuss if the literate programming paradigm is appropriate for object-oriented languages. The literate programming paradigm excels in the documentation of large complex systems, but do we need literate programming when using OOP? There's no doubt that documentation is a good thing but is the overhead of scraps, etc. needed to document a system consisting of the small components promoted by OOP?

I've been doing quite a lot of work on a system in Dylan lately. (For those who don't know about Dylan, you can think of it as a very fast CLOS with algebraic syntax and optional strong typing.) I don't think that object-orientation does anything to reduce the value of the literate programming approach. In my use of object-oriented languages, I've found that programmers make far too many assumptions about how obvious things will be. Each little tiny element of code makes sense on its own, but its place in the complete system is far more difficult to understand. So what I end up doing in object-oriented literate programming is to provide a lot of documentation about the *system*, and less about the components. I use a heirarchical structure that matches the basic structure of Dylan programs.

In Dylan, you build code in libraries. Each library consists of a collection of modules, and a declaration that imports modules from other libraries, and exports modules to other libraries. Each module is a private namespace, with imports of identifiers from different modules, and exports of identifiers to different modules. The modules consist of declarations of classes, methods, and variables. So I'll start on the top, and provide a description of the design of the whole system that I'm building, and how it's divided into libraries. If I've got a system diagram that makes sense at this level, I include it.

Then I provide the library declarations, with documentation about what services the library provides to the system, and how its divided into modules. (The library declaration provides a root chunk for one code file.) Each library is a chapter in the overall document. Then I write a section for each module. The module descriptions are less ordered and structured than the high levels, depending on what order of presentation will make the most sense to a reader. Then there's the chunks describing the different classes and methods. The methods are not usually broken into more than one chunk, except in the relatively rare cases that they're very complicated.

In my view, documentation for an object-oriented system should consist of: 1. Description of the relationships between the classes/class clusters. Here a suitable modeling technique should be used; e.g. Rumbaugh and/or Booch. 2. How to use the class and each of its methods. 3. A description of the algorithm in each method, but normally there will be no need to split the method into more than one scrap. We can meet these requirements without using the literate programming paradigm.

You can do that without using literate programming. But similarly, I can write a program in C which well structured and documented without using literate programming. The trick is that it's a lot harder to keep it well documented when the code and the documentation are seperate. Especially when someone changes something in a hurry, and doesn't alter the documentation to match. That's a lot more likely to happen when the documentation is seperate.

From: **Jacob Nielsen**
Date: 16 May 1995

This section should, IMHO, go into the FAQ; especially the '2)' !

Marc van Leeuwen writes: Literate programming is not identical with the use of modules (scraps, chunks, ...). In fact these are just a means towards the goal of providing the highest possible level of documentation to the code. They are however very effective in the situation they are intended for, because they achieve two things at once: 1) move code dealing with details that are not discussed in the commentary of the current section outside of its view, so that the connection between the commentary and the (visible) code is more easily perceived; 2) provide a short abstract of the omitted code through its module name. The second point should not be overlooked; sometimes it is so effective that some sections need no commentary, because the module name already says it all (just like some mail messages need no body after stating their subject line). It is also a point that will be missed if one just uses function calls instead of module abstractions as a means of partitioning the code into small pieces; it is not likely that one will use function names with some 100 characters or so, even it the language allows it and the task of the function cannot be described well with fewer characters.

In my view, one should not make functions just to partion other functions into smaller pieces. Use a good editor (emacs springs to mind) or better still use the literate programming paradigm.

It seems to me that the distinguishing feature of the object-oriented paradigm might well lie not in the methodology itself, but rather in the class of problems that it can effectively be applied to. From my experience it seems that there are some problem areas, in particular event driven programming such as occurs in programs with an elaborate graphical user interface, where one is led almost automatically into an object-oriented approach. [...]

Just adding: When you need the application to behave in a dynamic manner (e.g. load and execute different code modules on demand), OOP also makes it a lot easier.

However, I have never seen an example of a task of any algorithmic complexity (in the every-day sense of the word complexity) that could be handled effectively by object-oriented techniques;

Modern DTP programs? FEM (Finite Element Method) programs applied to soil mechanics? (many different types of soil with different characteristics)

I have really no idea how one could apply these techniques to make programs such as TeX more transparent, or (for a less monolithic example) programs like those in the Stanford GraphBase. My own experience, which ranges form writing mathematical software (computer algebra) and parts of a compiler to some work on a program with an extensive user interface, suggests that the utility of literate programming techniques increases strongly with algorithmic complexity: [...] But regardless of the algorithmic complexity, every large project will need an introduction in pure commentary that explains the task that is being performed, and the global structure that underlies the design; such a description is much more readable on typeset pages than as a huge block comment in the code.

Agreed! I too prefer typesetting large portions of text so I wouldn't put introductions etc. into a huge comment block.

I don't understand what you mean by a modeling technique in point 1.

The Rumbaugh and Booch modeling techniques use boxes etc *and* text to display/reveal the relations between different parts of a system; just like state diagrams, ... The interesting thing about these techniques is that they make use of several different kinds of diagrams and also encourages the use of several diagrams of the same classes/objects/whatever at different levels of abstraction.

From point 3 it seems to me however that the software you are considering has a very limited algorithmic complexity.

You're correct that I thought of small methods (~ scrap size), but just because one uses many small methods/functions (as promoted by OOP) with limited complexity, it doesn't follow that the system in full is of limited complexity.

From: **Lee Wittenberg**
Date: 16 May 1995

Jacob Nielsen writes: In a desperate attempt to outnumber the off-topic posts in comp.programming.literate, I want to discuss if the literate programming paradigm is appropriate for object-oriented languages. The literate programming paradigm excels in the documentation of large complex systems, but do we need literate programming when using OOP? There's no doubt that documentation is a good thing but is the overhead of scraps, etc. needed to document a system consisting of the small components promoted by OOP?

Absolutely. I find that literate programming is extremely useful for object-oriented work. Regardless of paradigm, most experienced programmers (at least those I've worked with) tend to write programs in little pieces and then put the pieces together to make the compiler happy (which is what the tangling tools do automatically). OOP reduces the amount of juggling you have to do by using objects for encapsulation, but there are still (and will always be, IMO) forms in the language that cater to the compiler at the expense of the way in which humans think. The main purpose of literate programming, as I see it, is to enable the programmer to write a program for a human reader, yet still enable a compiler to grab hold of it. Aside from writing programs in natural language (and I don't want to get into that sidebar), the best way seems to be the literate programming chunk technique. As Dijkstra says in "Structured Programming", "I want a program written down as I can understand it. I want it written down as I would like to explain it to someone." That's precisely what literate programming tools provide.

For example, when I develop a class, I always lay out the outline (boilerplate) first:

```
class X {
  @<|X|'s private members@>@;
protected:
  @<|X|'s protected members@>@;
public:
  @<|X|'s public members@>@;
};
```

This way, I can describe the members themselves in whichever order makes best sense for the human reader. Frequently, while discussing one of the public members, I'll need to explain a private member to make my meaning clear (or at least clearer). [Note: I use C++ and CWEB in this example because I have to use *something*. This does not constitute product endorsement of any kind. I want to avoid religious wars of this sort.]

1. Description of the relationships between the classes/class clusters. Here a suitable modeling technique should be used; e.g. Rumbaugh and/or Booch.

Reasonable, but this does not preclude its inclusion in the web.

2. How to use the class and each of its methods.

This is more user documentation rather than programmer documentation. On the other hand, I've had quite good results with including manual pages as a detachable part of the web (Norman Ramsey does it even better than I do), so the user docs and the program stay in sync. Again, literate programming is helpful.

3. A description of the algorithm in each method, but normally there will be no need to split the method into more than one scrap.

I disagree here. When I'm working with an algorithm I write myself, I find that I now use chunk names in place of pseudo code, and the chunk definitions in place of refinements. Any nontrivial algorithm, IMO, will require a not insignificant amount of stepwise refinement. In addition, the text chunks allow me to reason explicitly about my algorithm and prove (at least informally) that the algorithm is correct. I can then demonstrate that the code is correct by showing that it matches the text description. On the other hand, when I already have an algorithm (from a book, perhaps), I find that literate programming allows me to translate that algorithm almost directly (and therefore correctly). I have an example program that uses Knuth's topological sort algorithm in just this way, but I don't have it readily available.

We can meet these requirements without using the literate programming paradigm.

True, but we can do any programming without literate programming techniques. Literate programming simply enhances the readability and maintainability of our code, regardless of which paradigm we use.

I have used literate programming in a C++ project and really enjoyed the ability to introduce instance variables when they were first used in a method, but the need and reason for these variables should have been discovered and documented in the analysis phase. My need for scraps were, more or less, limited to containing a method in full. In hindsight, the project could have been documented equally well using good commenting practices and suitable modeling techniques only.

I disagree with this paragraph entirely. If programmers were perfect, your statement about the analysis phase would probably be correct, but we're not, and we can't always foresee the future with perfect clarity (even with respect to program development). Similarly, I think there is a significant difference between standard commenting practices (however good) and literate programs. A program has 2 audiences, the human reader and the compiler, who have different needs. Standard comments make a compromise that loses the ability to use exceptionally human-related explanations (such as charts, graphs, indexes, etc.) in order to allow the documentation and code to coexist in the same file. Literate programming makes that compromise unecessary.

I doubt that I will use a literate programming tool in future projects involving OO-languages but I will most certainly document my work to a much larger extent than before I learned of literate programming and realized just how much was missing from my previous documentations.

I'm sorry to hear that, but of course, each of us must work as he (or she) thinks best. I suspect, though, that you will find that, even with the best standard documentation you can do, you will still miss literate programming.

From: **Peter Horan**
Date: 17 May 1995

Literate Programming wins when the issue is algorithmic complexity. Object-oriented programming (or indeed, Structured Design) wins when the issue is system complexity. Design the system properly. Marry the methods if required by the complexity of the code.

From: **Mike Elliott**

Date: 17 May 1995

Jacob Nielsen writes: The literate programming paradigm excels in the documentation of large complex systems, but do we need literate programming when using OOP? There's no doubt that documentation is a good thing but is the overhead of scraps, etc. needed to document a system consisting of the small components promoted by OOP?

I think the view of many is that literate and OO programming are basically orthogonal -- both have benefits and can be usefully combined. However, if one is to adopt the view of any of several OO design methodologists such as Grady Booch or Jim Rumbaugh, there are aspects of OO design which have no direct implementation in any existing language: concepts such as Booch's class category. Literate programming offers a method of implementing some of these design abstractions (such as the class category) directly -- even without regard to the underlying OO language. I therefore feel that there exists a synergy between the two techniques that is greatly under-utilized.

From: **Phil Jensen**
Date: 18 May 1995

Marc van Leeuwen writes: However, I have never seen an example of a task of any algorithmic complexity (in the every-day sense of the word complexity) that could be handled effectively by object-oriented techniques; I have really no idea how one could apply these techniques to make programs such as TeX more transparent, or (for a less monolithic example) programs like those in the Stanford GraphBase.

Hear, hear! I have felt this very strongly. The religious fervor of the OO true believers seems utterly misplaced to me: there are only a few new ideas there, and (as Marc said earlier in his article) they only really shine in a few problem areas, ones which are uninteresting to me. Of course, they do harmonize with the current belief that computers primarily draw pictures.

My experience with literate programming

From: **Greg Humphreys**
Date: 14 May 1995

In an effort to spark some sort of thread that is on the topic, I'll tell you all why I love literate programming and people can share their success / failure stories. How many times has the following happened to you: You sit down to write a non-trivial program for either a class or work or enjoyment or whatever, and it turned out to be several hundred lines long. It took you about 2 hours to write, and then you sit to try to debug. That takes about 2 days because of some silly little thing that you didn't even notice.

Used to happen to me all the time. Then, for my systems programming seminar class this semester, our prof. insisted that we use noweb for all our programs. The first three programs (!) which were: Generating efficient Mips assembly code from the switch statement, a Threads package, and Multiple precision long division, all worked upon compilation. That NEVER happens. EVER. I attribute this success story to literate programming entirely. In my opinion, it's a lot harder to get 10-20 lines of code wrong if you write a paragraph describing exactly what they are supposed to do, and if those 10-20 lines are some logical chunk of your program that the language would prevent you from separating out.

In addition, when your program is done, you have 30-65 pages of TeX output ready to hand in (for this class, anyhow)... no writeup! Being a big TeX advocate, [[noweb]] has been like a dream come true. Recently I started switching to |CWEB|, but I'm not sure I like the output. With [[noweb]], I could have complete control over the sectioning of the LaTeX document, but CWEB has its own ideas about paragraphs and sections and tables of contents that don't really make me happy.

Here's a question for all you literate programming fans out there who are dying to respond to some literate programming related post: Is there a literate programming system (for unix) that pretty prints your C and allows you to do things like |CWEB| does with your output, yet still gives you control over the sectioning like [[noweb]]? I liked noweb because all it allowed me to do was to break my code into chunks and document it on the fly, but I like the way the code in |CWEB| looks better. What's my tool of choice?

Hope this can at least provide some thread of interest for readers of this group. Anyone have failure stories about literate programming? I want to hear them (so I don't repeat your mistakes :)). I plan to write every piece of non-trivial code from now on in some literate programming system.

From: **Marc van Leeuwen**
Date: 15 May 1995

Greg Humphreys writes: Here's a question for all you literate programming fans out there who are dying to respond to some literate

programming related post: Is there a literate programming system (for unix) that pretty prints your C and allows you to do things like |CWEB| does with your output, yet still gives you control over the sectioning like [[noweb]]? I liked noweb because all it allowed me to do was to break my code into chunks and document it on the fly, but I like the way the code in |CWEB| looks better. What's my tool of choice?

This has been on my non-urgent wish list for some time, together with a couple of other CWEAVE variants. While I probably won't get do it on the short term, here is the simple idea that should do the trick: add just one code to the (C)WEB langage that signals the end of a section without starting a new one, but rather gets you into a similar mode as you had in limbo (i.e., not in any section, but talking directly to TeX). That would give you all the opportunity you need for sectioning (in the document sense, rather than the WEB sense) and other non-program related stuff. For a really pleasant result you might want some additional modifications, like changing the support macros that are currently geared towards the ``WEB-section as basic structuring element'' philosophy, and adding hooks to allow the WEB-section to be numbered subsidiary to the larger document sections. One step further would be to modify CWEAVE into an eqn-style filter that just prettyprints code sections, while passing everything else through unchanged (anybody volunteer to do this? :-).

From: **Joachim Schrod**
Date: 16 May 1995

Greg Humphreys writes: Here's a question for all you literate programming fans out there who are dying to respond to some literate programming related post: Is there a literate programming system (for unix) that pretty prints your C and allows you to do things like |CWEB| does with your output, yet still gives you control over the sectioning like [[noweb]]? I liked noweb because all it allowed me to do was to break my code into chunks and document it on the fly, but I like the way the code in |CWEB| looks better. What's my tool of choice?

CWEB & LaTeX. :-) Seriously, in the current state of affair you need some knowledge of (La)TeX programming. CWEB actually provides a five-level-hierarchy for sections. Chunks (started with `@ ') may not necessarily marked with numbers, it's possible to do without. Of course, then one has to tag the refinement definitions with numbers.

CWEB.sty provides an internal (programmers') interface that gives access to this document structure. The current default implementation produces the result as known from the plain TeX macros. For somebody halfway fluent in LaTeX coding, it's easy to set up a different rendering by writing a style option. Actually, one of the reasons that a LaTeX 2e CWEB document class hasn't been released yet is my wish to write such a package as an example. (The 2nd reason is that I don't use CWEB currently. ;-) PS: Of course, one can do the same in plain TeX. The problem you mention isn't a CWEB problem, it's a missing feature in Knuth's macros.

Marc van Leeuwen writes: Here is the simple idea that should do the trick: add just one code to the (C)WEB langage that signals the end of a section without starting a new one, but rather gets you into a similar mode as you had in limbo (i.e., not in any section, but talking directly to TeX). That would give you all the opportunity you need for sectioning (in the document sense, rather than the WEB sense) and other non-program related stuff. For a really pleasant result you might want some additional modifications, like changing the support macros that are currently geared towards the ``WEB-section as basic structuring element'' philosophy, and adding hooks to allow the WEB-section to be numbered subsidiary to the larger document sections. One step further would be to modify CWEAVE into an eqn-style filter that just prettyprints code sections, while passing everything else through unchanged (anybody volunteer to do this? :-).

CWEB.sty has all the code already. `Just' redefine the appropriate macros from the protected level (\CWEB ...), there are (IMHO) enough hooks.

From: **Werner**
Date: 16 May 1995

Marc van Leeuwen writes: This has been on my non-urgent wish list for some time, together with a couple of other CWEAVE variants. While I probably won't get do it on the short term, here is the simple idea that should do the trick: add just one code to the (C)WEB langage that signals the end of a section without starting a new one, but rather gets you into a similar mode as you had in limbo (i.e., not in any section, but talking directly to TeX). That would give you all the opportunity you need for sectioning (in the document sense, rather than the WEB sense) and other non-program related stuff. For a really pleasant result you might want some additional modifications, like changing the support macros that are currently geared towards the ``WEB-section as basic structuring element'' philosophy, and adding hooks to allow the WEB-section to be numbered subsidiary to the larger document sections. One step further would be to modify CWEAVE into an eqn-style filter that just prettyprints code sections, while passing everything else through unchanged (anybody volunteer to do this? :-).

You can also try my c2CWEB program. It will prettyprint ordinary C code, and in the usual case you just need to insert a few specially formatted C comments (something like /*@@*/ etc.). It uses a modified CWEAVE to print the source code together with a kind of

preprocessor which translates the C code into a CWEB input file. (I'm currently working on a different package, thus it is not adapted to Leeuwen's CWEBx which I prefer).

From: **Lee Wittenberg**
Date: 16 May 1995

Marc van Leeuwen replies: This has been on my non-urgent wish list for some time ... One step further would be to modify CWEAVE into an eqn-style filter that just prettyprints code sections, while passing everything else through unchanged (anybody volunteer to do this? :-).

A "Spidery noweb" toolkit for building noweb filters to prettyprint code chunks has been on my "to do" list for a long time. Once I can find the time (a major problem, as we all know), I intend to build at least a prototype and distribute it. However, until that time comes, I am only a tentative volunteer. If anyone else wants the job (and has the time), I'll be glad to pass along my ideas.

From: **Will Ware**
Date: 16 May 1995

Greg Humphreys writes: In an effort to spark some sort of thread that is on the topic, I'll tell you all why I love literate programming and people can share their success / failure stories.

As long as people are sharing their literate programming experiences, I wanted to pass along mine. It isn't yet what I consider a resounding success story, but it's gradually getting there. A couple of months ago, I stumbled across literate programming. I don't remember if this was on the net, in a library, or a magazine article, but shortly I began voraciously reading whatever I found. It looked very useful, given that I and another engineer share responsibility for a large software project. I found Knuth's book and the literate programming FAQ, located some FTP directories, etc.

I wasn't familiar with TeX, and didn't know an easy way to get TeX output printed on any of the printers where I work, so I hunted for tools that could work without TeX. I was also concerned about the cryptic appearance of the web files I was able to find.

Simultaneously I began tinkering with my own tools. Initially I decided to create my source documents in Microsoft Word (a decision I have since reversed). One of the problems with that approach was that tangling required a Word macro, which couldn't be put in a makefile. The upshot of that effort was that I was able to get a bunch of code running for a demo, and show my boss and coworkers a very pretty collection of Word documents, and people had the unusual experience of actually understanding the code.

Then it came time to merge the code with somebody else's code, maintained on another computer, and what ended up happening was that after extracting all the .C and .H files to their computer, we never bothered updating the Word documents as we worked on the code. No tragedy ensued from this, but I concluded that I should use plain ASCII files for my source documents. Some reasons for preferring an ASCII source document are (1) much quicker and easier to edit, (2) much smaller, (3) works well with standard programming tools like AWK and GREP, (4) no mystery Microsoft formatting, (5) version control systems like ASCII files.

I also began reading up on Postscript, since we have a very pretty Postscript laser printer at work. I found a program called Ghostscript that allowed me to print Postscript files on my 24-pin dot matrix printer at home; they actually look pretty good, albeit slow.

I had initially wanted to avoid using any specialized language to control tangling and weaving, but I ended up with a language of suitable simplicity. It has only nine '@' commands. I don't even need a cheat sheet. So the tools I ended up with have the following pleasant features: the source document is plain ASCII, the TANGLE and WEAVE processes can both be put in a makefile, and the output is a Postscript file suitable for printing at home or at work. I was able to update all my Word files, and convert them to the new system, in about a day.

This probably sounds like a pretty round-about path just to avoid TeX. But it was highly instructive, and I saved a lot of hard disk space and hassle by avoiding TeX and dvips. The tools I ended up with are quite compact, since all the really hard work is done by either a Postscript printer or by Ghostscript.

From: **David Thompson**
Date: 17 May 1995

Greg Humphreys writes: In an effort to spark some sort of thread that is on the topic, I'll tell you all why I love literate programming and people can share their success / failure stories.

I'll briefly comment on this. First, thanks for posting something on topic. ;-) I've been literately programming now for a couple or three years. I started by dabbling with cnoweb and a simple example program, then moved on to more serious literate programming systems (and I do know the difference). I've been using both noweb and fweb recently, and have had an ongoing discussion with

Norman Ramsey over literate programming and (of all things) Fortran. I find that I much prefer the degree of control over formatting given by noweb. However, it's much easier to handle the output of the Fortran source using fweb. It's a quandry for me as to which tool I'll focus on. However, for now, it will be FWEB until I can formulate a post-processor for noweb output that will clean up the Fortran source into some compiler-acceptable form. My programs aren't perfect; but they get the job done. And, I can go back to them after some time and pick up the pieces. That says quite a bit about the paradigm.

From: **John Haxby**
Date: 19 May 1995

Greg Humphreys writes: In an effort to spark some sort of thread that is on the topic, I'll tell you all why I love literate programming and people can share their success / failure stories.

*** Success: I wrote a device driver some time ago using noweb. The final document included not only the C code for the driver, but the shell scripts needed to install and uninstall it, an example program and the Makefile to build the driver and format the document. [Actually, there were two documents, a README describing how to extract the Makefile and use it and the literate program document itself...] There were few choices I could make over the use of the literate programming tool since I wasn't confined to a single programming language. The success came when I was reviewing the document for typos and whatnot (after the driver had been written and tested) -- I found that the mechanism for protecting critical sections could be done better, so I changed the <<sleep>> and <<wakeup>> chunks, and I also found, as a result of seeing the program uncluttered by device-driver housekeeping, a small but significant bug. Oh, and of course, the nicely formatted document provides detailed and accurate documentation for both the driver software and the hardware (the hardware documentation proved especially elusive when I was writing the driver in the first place...)

** Failure: I started writing a Tcl/C++ program using literate programming, but it rapidly fell apart when I realised that I didn't have a good design to pin the document structure to. Every time I wanted to modify the design, I had to more-or-less re-write the literate program document to reflect the new design. I went through two design phases like that before I quit using literate programming. That was the prototype, the real thing, if it ever gets written, will have a proper design and it will be well-worth using literate programming, now I know what the structure of the document should be. Moral: get the design right first.

From: **Bakul Shah**
Date: 21 May 1995

John Haxby writes: I started writing a Tcl/C++ program using literate programming, but it rapidly fell apart when I realised that I didn't have a good design to pin the document structure to. Every time I wanted to modify the design, I had to more-or-less re-write the literate program document to reflect the new design.

What sort of tools would have helped this process of discovering a good design? How did your choice of languages (Tcl/C++) affect the design space you wanted to/were able to explore? Discovering a good design is the hardest, the most frustrating and the most enjoyable part of programming for me. I try to manage this activity by writing design notes along with code being developed. When the design changes radically I clone a new directory and make changes there. But this has not been very satisfactory. How do other people manage/document design explorations?

Most all literate programming uses I have seen are for finished programs but it seems to me that it can be useful during program development as well. I'd like to be able to document various design branches taken, rejected or explored, extract a final document, extract a historical perspective, etc. Any thoughts on how to go about this?

A related question is how to manage literate programming in a multi-person project. Typically people produce design documents, using TeX or some word processor, that frequently get out of sync with the related code. Again, the problem is managing the program development process (and I don't mean use of RCS or some such source code control system).

From: **Stephen Mc Kearney**
Date: 22 May 1995

Bakul Shah writes: Discovering a good design is the hardest, the most frustrating and the most enjoyable part of programming for me... Most all literate programming uses I have seen are for finished programs but it seems to me that it can be useful during program development as well. I'd like to be able to document various design branches taken, rejected or explored, extract a final document, extract a historical perspective, etc. Any thoughts on how to go about this?

What attracted me to literate programming was the very enthusiastic comments at the start of Knuth's writings on the subject --- programming is like writing a book. I take this to mean that the programmer writes drafts, outlines, etc. Throws them away when they don't work and starts again. Finding the right design is about finding the right document structure. I can't say that this is how I write programs (I don't write many) but listening to this newsgroup I get the feeling that people use literate programming as a way to reorganise their traditional programming methods. Does anyone use this very literal (and slow?) approach to literate programming?

You presumably have to be a very good writer and programmer.

From: **Lee Wittenberg**
Date: 23 May 1995

Bakul Shah writes: Most all literate programming uses I have seen are for finished programs but it seems to me that it can be useful during program development as well. I'd like to be able to document various design branches taken, rejected or explored, extract a final document, extract a historical perspective, etc. Any thoughts on how to go about this?

I find literate programming quite useful during program development, as well. While in the "noodling around" phase, I generally write explanations in a "stream of consciousness" fashion (as ideas occur to me), inserting code chunks whenever I describe something that should be code. When I've got this rough draft finished, I weave (but never tangle or compile) it, and use these thoughts as a basis for a "real" design. Usually, I find that I use most of the code chunks (and their associated explanations), but the final web has them in a completely different order (geared to the reader), and often in a totally different program structure, as well.

The greatest benefit of literate programming, IMO, is that a programmer's thoughts no longer disappear into thin air once the program is written; they are preserved in the web. The maintenance programmers who come after have these thoughts as a foundation for future work. These thoughts, as you say, should include "branches taken, rejected or explored". This saves later programmers from following trails known to be false.

Stephen Mc Kearney writes: What attracted me to literate programming was the very enthusiastic comments at the start of Knuth's writings on the subject --- programming is like writing a book. I take this to mean that the programmer writes drafts, outlines, etc. Throws them away when they don't work and starts again. Finding the right design is about finding the right document structure. I can't say that this is how I write programs (I don't write many) but listening to this newsgroup I get the feeling that people use literate programming as a way to reorganise their traditional programming methods. Does anyone use this very literal (and slow?) approach to literate programming? You presumably have to be a very good writer and programmer.

I tend to believe that I work that way. In any event, I've always been a "slow and steady" programmer, even before literate programming. As I mentioned in earlier messages, I often treate webs as drafts and completely reorganize or rewrite them. I also suspect that a great many of the literate programmers out there do pretty much the same thing. I guess we use "programming talk" rather than "writing talk" because we're thinking more about programming than writing (the final result has to work, after all, not just look pretty).

Moderate comp.programming.literate

From: **Will Ware**
Date: 14 May 1995

Don Hosek wrote: [comp.programming.literate should be moderated]

That sounds like a bad idea to me. My small amount of experience with moderated newsgroups is that moderation reduces the volume even further (perhaps because people enjoy seeing their words made public immediately). A good deterrent to off-topic posts might be if there were a high volume of on-topic posts, so that potential posters would have a chance to see what this newsgroup is nominally about.

Thinking back, I can remember two general flavors of on-topic posts that I've seen in this newsgroup. Most of them are: 1. Discussing the details of getting some literate programming tool to work with some language, or on some machine, or other environment-related stuff. Almost all the rest are: 2. Asking (or responding) about where to find web pages, ftp sites, etc. pertaining to literate programming.

Those are good and useful things to discuss, but perhaps there could be more kinds of posts. In many newsgroups about topics that people feel strongly about, there are sometimes a few posts which: 1. Express in extreme (albeit usually pretty inarticulate) language the author's wild enthusiasm for the topic (in this case, literate programming). Too many of those would be annoying, but a small volume of them might make it easier for the reader to re-ignite his or her own enthusiasm. As somebody who is pretty new to the literate programming paradigm, I'd appreciate seeing: 2. Success stories ("I tried literate programming for this, and it was just the cat's pajamas!") 3. Horror stories ("I'll go to my grave regretting the day I decided to use literate programming for this project!")

As a newbie who hasn't yet settled on one of the standard literate programming tools, I am still experimenting with some ideas of my own. (For instance, I don't like TeX much, so I'm playing with Postscript output. I know, TeX and Postscript deal with entirely different

levels of abstraction in preparing documents, but I'm getting output that I'm pretty happy with.) Maybe there could be posts about: 4. Experimental sets of literate programming tools.

Similarly, there might be posts about topics that were somewhat related to literate programming, but not specifically about literate programming: the kind of thing that would interest somebody who was also interested in literate programming. Examples might be new tools for document processing, or new programming languages that tried something new in order to be more self-documenting, or features of HTML that could be exploited in a hypertext literate programming tool: 5. Related topics.

Moderation would reduce the volume of a newsgroup that is already gasping for air. It would be better if the conversation about literate programming were so wildly flourishing that even with off-topic posts at the same rate they are today, it would be hard to find them in all the on-topic hubbub. One could even imagine a newsgroup so alive and vibrant that it was responsible for a wave of new literate programmers.

I remember two other ideas that sounded very good. One was to post a FAQ every week by some automatic means. The other was to change the newsgroup's description so that the topic is more obvious to somebody not already familiar with literate programming.

From: **David Kastrup**
Date: 15 May 1995

Don Hosek writes: Perhaps this is the answer. Among the off-topic posts on this pass-thru: an HTML question and an anti-mac flame. Volume is low so it wouldn't be a great burden to switch to moderation. But no, I'm not volunteering for the task.

Another possibility would be to switch the misleading (to those not in the know) group name. comp.programming.lp, or comp.programming.lp-systems, or comp.programming.webs or similar come to mind. However, I think a moderated group will have the advantage of attracting more people to literate programming, which just happen to chime in.

Another possibility would be pseudo-moderation: an automatic moderator (augmented by humans of course). Anyone not "registered" will get his posts sent to the human moderator. In addition, he is sent the FAQ automagically. After an explicit mail to the automatic moderator with the line "I have read the FAQ" (which, of course, is mentioned in the FAQ), the user gets registered for the automatic moderator. Why this fuzz? Because moderation stifles discussions, as the moderation introduces delays. The above technique would solve the problem.

Now who'll write the software, and who'll volunteer for human moderator (because I think it will be impolite to just drop a post because it might possibly be off-topic). Another solution would be that you could include a sentence "Please post my previous post" in the "I have read the FAQ" mail to the automatic moderator, or "Please cancel my previous post" (which gets done after a week or so automatically). This solution would more or less stop the necessity for human intervention, except for for perhaps entering obnoxious "persona non grata", which cannot automatically post anymore.

From: **Lee Wittenberg**
Date: 16 May 1995

The problem seems to be that the newsgroup description at AOL is wrong. And nobody I've been in contact with was able to tell who's responsible about that wrong description.

I promised myself that I wouldn't get involved in this discussion, but I guess I'm going to break that promise. I've always been under the impression that the reason we've been getting all this junk is that people have been posting all of their programming questions to comp.programming.*, and that the comp.programming.scheme people are as upset about the general C++ and Visual Basic questions as we are. Is this not a viable theory (albeit without a solution)? Perhaps the only solution is to give up the newsgroup and go back to the Listserv.

Beginner's guide?

From: **Kumaran Santhanam**
Date: 17 May 1995

I found the FAQ and hunted around for all of the documents I could find to get started with literate programming, but still have yet to find a good step-by-step beginner's document to programming under a system such as noweb, for example. Can somebody please point me to an on-line beginner's tutorial which would allow me to learn literate programming with a minimum of pain?

From: **David Thompson**
Date: 19 May 1995

Kumaran Santhanam says: Can somebody please point me to an on-line beginner's tutorial which would allow me to learn literate programming with a minimum of pain?

So far as i know, no such document exists. I learned by taking a tool (noweb is a good place to start) and started writing programs. Of course, I had been programming for a good many years, and was familiar with latex (use it all the time), so I was able to focus on learning to use the tool without worrying about those other issues. A good place to start might be to take one of the noweb examples and study it, but in source form and after typset and printed. This should give you a fair amount of data to start from.

From: **Kumaran Santhanam**
Date: 19 May 1995

David Thompson writes: So far as i know, no such document exists. I learned by taking a tool (noweb is a good place to start) and started writing programs. Of course, I had been programming for a good many years, and was familiar with latex (use it all the time), so I was able to focus on learning to use the tool without worrying about those other issues. A good place to start might be to take one of the noweb examples and study it, but in source form and after typset and printed. This should give you a fair amount of data to start from.

I already did that, and I think I'm starting to get a handle on it. I also have been programming for a long time, and I use LaTeX as well, so focusing on learning noweb was no problem. I think that after I write a couple of programs in literate form instead of in traditional C, I'll get the hang of it...

From: **Bill Rassieur**
Date: 22 May 1995

Your stated goal is to find out what the jist of literate programming is... May I suggest (invite) that you try it? 1) Read a WEB somebody has written. 2) Write a program as a WEB. In the case of (1) at the very least you should be able to locate any of a number of books that Donald Knuth has authored that are, in essence, Webs (The Stanford Graphbase... CWEB... sorry, I don't recall exact titles ;-) In the case of (2) download CWEB, noweb, of any of I don't know how many WEB tools from the net... (you only need 1). And congratulations for posting "on topic!"

From: **Guy Bailey**
Date: 22 May 1995

Bill Rassieur writes: In the case of (1) at the very least you should be able to locate any of a number of books that Donald Knuth has authored that are, in essence, Webs (The Stanford Graphbase... CWEB... sorry, I don't recall exact titles ;-)

Most folks are unwilling to spend money unless they absolutely have to! :) But if you can spend a small amount of money, then I recommend Wayne Sewell's Weaving a Program: Literate Programming in Web, Van Nostrand Reinhold, NY, 1989. ISBN 0-442-31946-0. Although the book directly addresses Knuth's original WEB system (which used Pascal), the book contains a very good introduction to writing literate programs and using the WEB system which should apply to most literate programming environments. In addition the book briefly touches on Levy's CWEB and Ramsey's SPIDER.

From: **Graham Lyford**
Date: 26 May 1995

I have been listening to the group for a short time, and would like to say hello. I joined this group to find out if literate programming was a useful tool to aid in documenting programs or merely an academic toy (the jury is still out, I think I need to try one now). I am new to literate programming, and the internet, and would like to know where to start in becoming literate.

All my code is written on a PC, most of it cross-compiled for an embedded system. I have recently managed to get away from the local Forth language to work in C (boy did I enjoy doing that!). I am looking for a setup which will handle different languages (I still have to support Forth, and maybe C++ in the future) and will run on a PC. I welcome any advice and suggestions that the group can give me.

From: **Lee Wittenberg**
Date: 26 May 1995

Graham Lyford writes: All my code is written on a PC, most of it cross-compiled for an embedded system. I have recently managed to get away from the local Forth language to work in C (boy did I enjoy doing that!). I am looking for a setup which will handle different

languages (I still have to support Forth, and maybe C++ in the future) and will run on a PC. I welcome any advice and suggestions that the group can give me.

I suggest that you try noweb. The other language-independents (see the FAQ) are also okay, but I think noweb is the easiest to learn and the most flexible. If you decide to use the (La)TeX back end (you can also choose HTML, and I understand that plain ASCII is soon to be an option), I can heartily recommend Eberhard Mattes' emTeX distribution for the PC. You will also find Norman Walsh's "Making TeX Work" (O'Reilly, 1994) invaluable in helping you set up TeX on your system, whichever distribution you choose. I truly regret that Walsh's book was not available when I was setting up emTeX for the first time.

From: **Bart Childs**
Date: 26 May 1995

Start by looking at examples. There are several. Some very low level ones are maintained by me. More on that in a minute. What I will say now generally goes for all except MAC users. If a real emacs is now available for MACs, then it should go for them too. I know there are many reasons for NOT emacs. However, the reasons for outweigh them (IMHO). Get started by:

1) Get FWEB, emacs (preferably v 19 +), TeX, and web-mode.

2) Get most (or all) of the PostScript files from "ftp://www.cs.tamu.edu" directory "pub/tex-web/web/DOCs" (Incidentally, web-mode is in EMACS.web-mode at this same level).

3) Read IntroC and its companion NewtonC. Read wm_0 and wm_1, wm_2, wm_3, wm_4. In the last two I have omitted the .ps extensions.

4) If you can read this and have not been turned off, then read Knuth's book on Literate Programming. Read the preferatory material of the books TeX: The Program or METAFONT: The Program. Read Wayne Sewell's book recently mentioned in a post. JUST KIDDING, any one of these four is likely enough.

5) Inhale and write your first literate program. But don't do the program in the ususal way. Write about the program in much the same level of detail that is in wm_1.web (above). Iterate, add to it, get it reviewed, and have bunches of discussions (also know as arguments) as to whether it is done appropriately. This part is called peer review. If you don't have a peer, I will be pleased to review, discuss, ... within the limits of my finite capabilities. (I should also impose a limit of F90, F77, C, C++, and Pascal high level languages.)

The reason I said FWEB is that it supports F77, F90, C, and C++. If you do Pascal, you must get the original. We also find the emacs reference card (source and PostScript are there.) I would also be pleased to hear from those who try this and feel it is not appropriate. This is a subset of what we did in the courses mentioned in the previous post.

From: **Norman Ramsey**
Date: 08 Jun 1995

Bart Childs writes: 1) Get FWEB, emacs (preferably v 19 +), TeX, and web-mode. ... I would also be pleased to hear from those who try this and feel it is not appropriate.

I have tried this (outside a classroom setting) WITHOUT Bart's introductory literature (which I look forward to reading) with a number of people and the consensus is that some people flounder if just handed FWEB or CWEB. It is worth trying noweb or nuweb, which are much simpler. noweb and nuweb also support HTML, which means those who don't know TeX but do know HTML have that leg up.

From: **Bart Childs**
Date: 14 Jun 1995

Norman Ramsey writes: I have tried this (outside a classroom setting) WITHOUT Bart's introductory literature (which I look forward to reading) with a number of people and the consensus is that some people flounder if just handed FWEB or CWEB. It is worth trying noweb or nuweb, which are much simpler. noweb and nuweb also support HTML, which means those who don't know TeX but do know HTML have that leg up.

I have complete confidence that Norman is right that some flounder. Heck, I am a CS professor and I have seen thousands flounder. It is a fact of life. I am not sure about the second statement and let me first sound like a hypocrite. In a similar vein, my experiences causes me to believe it is better to learn Pascal before C for much the same reason that he espouses. I really think that when one learns literate programming, they are usually far enough along that learning a full one is worth it and besides, I just do not like monospace fonts, like the one this is in.

Why is literate programming nearly unused?

From: **Basile Starynkevitch**
Date: 18 May 1995

Why is literate programming nearly unused? For instance, nearly all published programs on the Internet (eg available by anonymous ftp) are *not* literate programs (eg most FSF or GPL-ed programs, such as GNU gcc, GNU emacs, Linux, are not literate). The only exceptions I know are TeX -famous text formatter at many CTAN ftp archive sites- and lcc -a C compiler, at ftp.cs.princeton.edu; the documentation is available as a book, @Book{Fraser:retargetable:book, author="Christopher W. Fraser and David R. Hanson", title="A Retargetable {C} Compiler: Design and Implementation", publisher="Benjamin/Cummings", address="Redwood City, CA", annote="compiler as a literate program", year=1995, note="ISBN 0-8053-1670-1" }

Is literate programming used on proprietary (or commercial) medium or large software (eg more than 100KLOC)? Only a few papers mention that. Norman Ramsey, in a paper 'Literate Programming Should be a Model for Software Engineering and Programming Languages' (march 1995), advocates that new languages should incorporate literate programming techniques, and that interactive editors should be made for assisting literate programming. I'm wondering why literate programming is almost never used? (Perhaps it is because most programmers are poor writers, and dislike batch formatters such as TeX?) Any comments?

From: **Mario Butter**
Date: 18 May 1995

I'm trying to get literate programming accepted here, but have not had much luck. One of the programmers on my team said that if I forced him use literate programming, he would never forgive me. I'm going to try to write a few sections of my next project using literate programming, and see how it goes over with management.

From: **Mary Bos**
Date: 18 May 1995

Is literate programming used on proprietary (or commercial) medium or large software (eg more than 100KLOC)? Only a few papers mention that.

The general software industry is only becoming aware of literate programming. As for the Gnu code, someone (or some group) needs to make an effort to make the package into a literate program. Literate code just doesn't happen in legacy code any more than all software is improved by software maintenance.

I'm finishing up my MSE (Master of Software Engineering) at Seattle Univ (in Seattle, WA) and still find literate programming has only be accepted by one person in my group of peers and professors, me! Most of the rest (especially those that work at a big software firm near Seattle) seemed to figure well commented code is enough, so why literate? These future software engineering managers aren't just ready to see the benefits yet. But then they don't do maintenance programming and management for a living and I do (and love it).

From: **Felix Gaertner**
Date: 19 May 1995

*Why is literate programming nearly unused? For instance, nearly all published programs on the Internet (eg available by anonymous ftp) are *not* literate programs (eg most FSF or GPL-ed programs, such as GNU gcc, GNU emacs, Linux, are not litterate). The only exceptions I know are TeX -famous text formatter at many CTAN ftp archive sites- and lcc -a C compiler, at ftp.cs.princeton.edu; the documentation is available as a book,*

This a question that is indeed very striking. If I talk to other students or go walking through the internet, nobody knows of literate programming at all. However, I know that a lot of advanced courses and projects at my university use literate programming tools and in the company that I worked with last year they used a CWEB variant to write really big programs too. So literate programming crops up at a lot of different places, but there are still so few that it's always a surprise.

I think it has to do with the way that people learn to program in school or at university. Although it might not be hard to switch from one programming style to the other, you have to have read a good literate program (or have had some other revalation) to really get into it. The first glance at a literate program is always confusing to traditional programmers.

When discussing this topic with fellow students I remarked, that literate programming is not so widespread because of its young age. But my counterpart said that the X windows system was even younger and almost everybody is using it today. What do people think

about that?

From: **Lee Wittenberg**
Date: 19 May 1995

Basile Starynkevitch asks: Why is literate programming nearly unused?

There are 3 reasons, IMO:

1. Most people don't know about it. The great bulk of writing about literate programming (aside from this group), has been in academic journals, and most working programmers, sadly, do not read them or even belong to organizations like the ACM or IEEE. A similar example is version control software. RCS and its ilk are used a great deal in academia, and in small pockets of commercial programming shops, but many software shops are ignorant of it, or

2. They can't be bothered. This is the same reason that very few programmers actually take the time to do structured programming. We all pay lip service to the Party Line, but when push comes to shove, we sit down at the terminal and hack away. Although we know in our heads that this leads to greatly increased debugging time, in our hearts we always believe that the program will work right the first time. Literate programming takes a great deal of effort to learn, and a web requires a great deal of thought up front, if you want to really explain your code to the reader. Even though this time spent up front leads to easier maintenance and reduced debugging time, the savings are ephemeral -- we don't really see them (how do you measure time you don't spend debugging?). The "savings" provided by rushing to the terminal (or skipping literate programming techniques) are immediate, and at least psychologically, measureable.

3. It's new and different. As Michael Crichton puts it ("Electronic Life", 1983): "Although we've all been fed a load of psychological pap that says learning is fun and challenges are exciting, human beings are inherently conservative. Change means stress. Change makes us uncomfortable and fearful.\footnote{Don't confuse change with newness. Newness is pleasant. You probably enjoy a new car or new clothes, but these new items are not really different from the ones they replace. As a matter of fact, automobile and clothing designers are careful to create the appearance of change without actually changing much -- because they know people don't like genuine change.}" Watch people learning to drive a car. Do they appear to be having fun? Of course not. They're terrified -- and dangerous besides. Watch people learning to play the piano. Are they having an exciting, challenging time? No. They're mastering feelings of frustration and boredom, trying to hit the damn keys right. There's a fundamental truth here: human beings don't like change.

A bit cynical, perhaps, and most of you reading this have already taken to literate programming (or are, at least, not afraid to try it), so these observations don't really apply to you (is this a cop-out? -- I don't think so, but like most human beings, I have a wonderful capacity for inconsistence). The hardest thing I had to learn when I started teaching was how a C student thinks. Never having been a C student, I was completely unprepared for dealing with them. My tirades above are against the "average" programmer (remember that a "C" grade is supposed to mean "average"), a group to which most of the programming population belongs (by definition?).

From: **Kristopher Johnson**
Date: 21 May 1995

Basile Starynkevitch writes: I'm wondering why literate programming is almost never used? (Perhaps it is because most programmers are poor writers, and dislike batch formatters such as TeX?)

Here are the reasons that I don't use literate programming on "real projects":

1) Lack of literate programming support in programming tools (editors, debuggers, profilers, etc.)

2) The output of (most) literate programming tools is TeX, which is not what I normally use for documents.

3) I use a Macintosh. Most literate programming tools were designed for a UNIX environment and are terrible on the Macintosh.

If someone would produce a *complete* literate programming environment which produced RTF, Microsoft Word, or WordPerfect documentation, I'd use it. With current tools, literate programming is too much trouble.

From: **Weiqi Gao**
Date: 21 May 1995

*Kristopher Johnson writes: If someone would produce a *complete* literate programming environment which produced RTF, Microsoft Word, or WordPerfect documentation, I'd use it. With current tools, literate programming is too much trouble.*

When I started out using literate programming tools, I tried CWEB, and FWEB. I can't use them because it takes too long to educate the next person in line to maintain my code. Now I use noweb, which is pretty easy to teach the next guy. The sticky point is TeX (or HTML) now. There is almost no hope to introduce TeX to a DOS/Windows/Word environment (we are talking about several hundreds

of Word people here). HTML might have a chance because of all the WinSock Internet apps. I haven't checked, but I think an RTF format output, which isn't that different from LaTeX, might be just what is needed here.

From: **Sven Utcke**
Date: 21 May 1995

Kristopher Johnson writes: Here are the reasons that I don't use literate programming on "real projects":

Followed by mine why I do use literate programming (more precisely: FWEB) on "semi-real projects" (research rather then development):

1) Lack of literate programming support in programming tools (editors, debuggers, profilers, etc.)

With cweb-mode, auctex and hilit in emacs, the gnu-debugger, gcc, FWEB and Insure++ I do have the perfect development tools at my hands.

2) The output of (most) literate programming tools is TeX, which is not what I normally use for documents.

TeX (or rather LaTeX) is what I use for all my documents (owing to the fact that as an engineer I have to use lot's of math. Not terrificly difficult one, but it still needs to be typeset...).

3) I use a Macintosh. Most literate programming tools were designed for a UNIX environment and are terrible on the Macintosh.

I use UNIX :)

However, the real reasons are: a) The ability to include images and math into my documentation (both about equally important, given that I'm working in computer vision). b) The possibility to use lot of pseudocode to write programs in exactly the way I'm thinking about them (rather then in the way my compiler thinks).

Having said all that, I have to admit that there is a HUGE drawback to literate programming in even semi-real projects --- which is the fact that most (well, all since last fall) of the people I work with do not use and do not intend to use literate programming. And the C-code produced by FWEB is not what I really want to share...

From: **Marc van Leeuwen**
Date: 22 May 1995

*Kristopher Johnson writes: If someone would produce a *complete* literate programming environment which produced RTF, Microsoft Word, or WordPerfect documentation, I'd use it. With current tools, literate programming is too much trouble.*

And Weiqi Gao writes: I haven't checked, but I think an RTF format output, which isn't that different from LaTeX, might be just what is needed here.

Unless I am missing something obvious here, aren't you both mistaken and really mean that you want a tool/environment that takes RTF, Word, or WordPerfect as INPUT rather than producing them as OUTPUT? I must assume that the output mentioned is that of the weaver (since the tangler has to produce compilable code, not RTF or other such format) but I really cannot see the point in manipulating this output with a WYSIWYG processor: the only useful thing to do with it is to get it printed (or displayed on the screen). If WYSIWYG processors have anything to offer to the literate programmer, it is in helping prepare the source files, not in tinkering with output files derived from them. It seems to me one would have to opt for one of the following possibilities:

1) Weaving is reduced to the identity operation. This means one has to forgo automatic prettyprinting, indexing, and such functions as numbering sections, except if such functions happen to be present in the word processor already. Effectively one edits the source by direct manipulation of the final document, and it is the user's responsibility to ensure the consitency of the document structure. To me this would seem extremely tedious, but at least such a literate programming tool is easy to implement (provided the document format is public), since one just needs a tangling program that can extract a program from a word processor document that has been set up according to the proper conventions (to identify code sections, module names, etc.)

2) Weaving is performed by an external program (or a sequence of them), followed by printing/previewing. To avoid elaborate processing during viewing and printing, the output format of the weaver should be close to the visual image; preferably dvi-format or postscript or some such. A format identical to the one (RTF, ...) used for input is conceivable here, but to use the same word

processor used for source editing also as previewer/printer driver would be hopelessly confusing. I do not know these word processors very well, but I doubt whether they have convenient means for running external programs and view or print their output during editing of the source file.

3) The weaving functionality is built into the word processing program. This would mean that the manufacturers have to be persuaded to incorporate literate programming support into their products (or to make their source code public, preferably as a literate program :-). This would be the most convenient solution, but it is not very likely to happen.

If I have not overlooked some attractive possibility, I think I may have given an argumentation why literate program using commercial text processors is not too popular. I would like to add a comment about using TeX as a formatter for literate programs: while this allows including math formulae and (with some standard support tools) illustrations into the document, one does not have to know very much about TeX when writing programs that do not use such frills; plain text will come out nicely, and only a few characters need a bit of special care. (Of course one does need a TeX installation for processing; maybe to promote the subject, somebody should piece together a minimal TeX configuration (no LaTeX please) for PC's that could be used for literate programming (only), and by a complete ignoramus.) Given that in illiterate programming the possibilities when using comments are severly restricted anyway, I don't think unfamiliarity with TeX should be a serious reason for not using literate programming.

From: **Ron Peierls**
Date: 22 May 1995

While all the barriers to widespread adoption of literate programming which have been suggested in this thread probably are significant, it seems to me that the major barrier is the lack of a single standard. As a result it is difficult, if not impossible, to share code even with other literate programming enthusiasts if they don't use the same flavor as you. If there were a standard even if it were at a low level with filters into and out of it for each WEB, then I think our crusades might be more successful. As it is I will remain within the FWEB stockade, even though there are probably some aspects of other systems which I would prefer.

From: **David Kastrup**
Date: 24 May 1995

Ron Peierls writes: While all the barriers to widespread adoption of literate programming which have been suggested in this thread probably are significant, it seems to me that the major barrier is the lack of a single standard. As a result it is difficult, if not impossible, to share code even with other literate programming enthusiasts if they don't use the same flavor as you. If there were a standard even if it were at a low level with filters into and out of it for each WEB, then I think our crusades might be more successful. As it is I will remain within the FWEB stockade, even though there are probably some aspects of other systems which I would prefer.

Well... In order to achieve that goal of easy interchangeability, the corresponding systems should adhere to a strategy of least unnecessary impact. This rules out systems which force a dedicated typesetting environment upon the text. Especially bad, in my opinion, are the many WEB variants *demanding* use of plain TeX. At least the use of the much more commonly used LaTeX should be supported without any special measures of circumvention. Taking some plain TeX hacker's private educational tool (the original TANGLE/WEAVE), and just marginally altering it will not leave the realm of plain TeX hackers, as much as I admire DEK. Among the users of TeX, the program, there is a vast minority of plain TeX users. Who'd want to learn a typesetting system not useful to much beyond the most basic typesetting purposes without further hacking? At least LaTeX has a lot of functionality built in.

In addition, the visual outlook of the source should be appealing. For proper debugging (unless you have access to tools making more clever use of the debuggers line information, like possible with gdb/emacs), this should be the case with unmodified ASCII source. Here noweb, in the mean, has advantages although the @<< convention (seldom needed) is rather ugly.

A particular disgrace in that respect is the mixing of typesetting stuff in the code (like @; or @t {1\over2}@> or whatever beauties various systems offer). This, for me, is reason to prefer non-pretty-printing systems, so that the non-typeset source looks cleaner. I think I'd even want to have a readable source if a clever combination of gdb, document style (generation of appropriate specials!) and xdvi would make it possible to step through a program using the typeset documentation, because debugging usually is connected with changing the source, recompiling and trying again, for which the source is handier.

In most academic outfits employing larger measures of literate programming, noweb was used, mostly because it was less intrusive than other systems, not because it was extendible, or modular, or had a nice syntax (the extensibility of noweb was nowhere used, and the modularity is not very interesting to the end user. And the nicety of the syntax is at least debatable). So in my opinion the greatest problem of literate programming are tools which are too clever, and require too much attention.

From: **Joachim Schrod**
Date: 24 May 1995

Felix Gaertner writes: When discussing this topic with fellow students I remarked, that literate programming is not so widespread because of its young age. But my counterpart said that the X windows system was even younger and almost everybody is using it today. What do people think about that?

Lewis Perin) writes: With all due respect, and even affection, for universities and computer science departments (I work for one of the former and graduated from one of the latter), anyone who thinks almost everyone uses X has seen only a narrow slice of the world of computing.

With all due respect, and even affection, for the other part of the world of computing -- may you please say something about Felix' argument, instead of quoting him in total and than adding this derogative remark? If you don't understand the original question, it's very simple: Let A == Literate Programming and B == X. B is younger than A. B is more complex than A. Nevertheless B is more widespread than A. Why had B such a larger success (measured in relative terms of visibility (e.g., media coverage) and usage) than A? The relationship between A and B have nothing to do with some system C that might be even more used than B. It's irrelevant.

From: **Lee Wittenberg**
Date: 24 May 1995

Weiqi Gao writes: There is almost no hope to introduce TeX to a DOS/Windows/Word environment (we are talking about several hundreds of Word people here). HTML might have a chance because of all the WinSock Internet apps.

All too true. This is precisely the reason I wrote the WinWordWEB prototype several years ago: to get WinWord people to at least *try* literate programming. The interesting result was that they tried it, but decided that noweb/LaTeX was better, so they now use noweb. Go figure.

I understand that WinWord 6 now supports hypertext and HTML output. If this is actually true, I'm going to try to get some money out of my school so I can make WinWordWEB into something more than a toy. If anyone has some spare grant money floating around and would like to support this effort (and save me the trouble of convincing my administration of the usefulness of this project), I will graciously accept (and get started that much sooner). :-)

I haven't checked, but I think an RTF format output, which isn't that different from LaTeX, might be just what is needed here.

RTF, while, TeX-like on the surface (it uses the backslash and curly braces in similar ways) is actually more troff-like in spirit (IMO).

Sven Utcke says: Having said all that, I have to admit that there is a HUGE drawback to literate programming in even semi-real projects --- which is the fact that most (well, all since last fall) of the people i work with do not use and do not intend to use literate programming. And the C-code produced by FWEB is not what I really want to share...

noweb has a lovely little tool called "nountangle" for just this kind of situation. You might want to take a look.

Marc van Leeuwen writes: If WYSIWYG processors have anything to offer to the literate programmer, it is in helping prepare the source files, not in tinkering with output files derived from them. It seems to me one would have to opt for one of the following possibilities:

1) Weaving is reduced to the identity operation. This means one has to forgo automatic prettyprinting, indexing, and such functions as numbering sections, except if such functions happen to be present in the word processor already. Effectively one edits the source by direct manipulation of the final document, and it is the user's responsibility to ensure the consitency of the document structure. To me this would seem extremely tedious, but at least such a literate programming tool is easy to implement (provided the document format is public), since one just needs a tangling program that can extract a program from a word processor document that has been set up according to the proper conventions (to identify code sections, module names, etc.)

The first statement is almost correct. As the (unfortunate) author of WinWordWEB, I can state for a fact that automatic prettyprinting, indexing and such are all possible in a WYSIWYG environment. In fact, Stephen Mc Kearney added some indexing features to the bare-bones system I produced. In WinWordWEB, prettyprinting macros would be straightforward, and non-intrusive, since the tangling function ignores typesetting codes in code chunks. The other alternatives would keep WYSIWYG types away in droves for the same reasons that they currently stay away from (La)TeX-based tools.

If I have not overlooked some attractive possibility, I think I may have given an argumentation why literate program using commercial text processors is not too popular. I would like to add a comment about using TeX as a formatter for literate programs: while this allows including math formulae and (with some standard support tools) illustrations into the document, one does not _have_ to know very much about TeX when writing programs that do not use such frills; plain text will come out nicely, and only a few characters need a bit of special care. (Of course one does need a TeX installation for processing; maybe to promote the subject, somebody should piece together a minimal TeX configuration (no LaTeX please) for PC's that could be used for literate programming (only), and by a complete ignoramus.) Given that in illiterate programming the possibilities when using comments are severly restricted anyway, I don't think unfamiliarity with TeX should be a serious reason for not using literate programming.

I agree. In fact the minimum amount of LaTeX knowledge needed to use noweb fits quite easily on a single page along with noweb markup symbols and the basic command-line information for notangle & noweave.

From: **Norman Ramsey**
Date: 25 May 1995

Basile Starynkevitch writes: Why is literate programming nearly unused?

Leaving aside from the usual NIH and similar problems, but I think there's a more fundamental problem: the payoff for writing literate programs seldom justifies the expense.

My colleagues and I tend to use literate-programming tools, but not to polish our programs (as readers of the noweb source know to their cost). We have found that the cost of polishing a literate program to a ``publishable'' standard ranges from 1 to 4 times the original development cost. The polishing process improves the code as well as the documentation, but the process is usually too expensive to justify. As researchers, we get paid to have new ideas and develop new technologies (and to publish about them), not to write beautiful programs. Unless a program is to be published as a book or a paper, making it literate is a waste of resources.

In a more production environment, things are no better---the marketplace does not reward quality in software (just consider the offerings of those who have made billions of dollars in software). The marketplace rewards time to market. For new-product development, literate programming is a way to lose in the marketplace---the company that ships first wins, even when their product is full of bugs. The one hope in all this is that literate programming could dramatically reduce maintenance costs and increase the useful lifetimes of products. There are a few companies who have experimented with literate programming and gotten these kinds of results. Unfortunately, much of today's software is built with a ``get it out now'' mentality, not a ``build it to last'' mentality.

Marc van Leeuwen writes: 1) Weaving is reduced to the identity operation. This means one has to forgo automatic prettyprinting, indexing, and such functions as numbering sections, except if such functions happen to be present in the word processor already. Effectively one edits the source by direct manipulation of the final document, and it is the user's responsibility to ensure the consitency of the document structure. To me this would seem extremely tedious, but at least such a literate programming tool is easy to implement (provided the document format is public), since one just needs a tangling program that can extract a program from a word processor document that has been set up according to the proper conventions (to identify code sections, module names, etc.)

Since Harold Thimbleby isn't here, let me argue on his behalf that this isn't really ``literate programming,'' since the correctness of the index and cross-reference isn't guaranteed by a tool. BTW, Lee Wittenberg took some steps in this direction with his WinWordWeb tool.

From: **Don Hosek**
Date: 25 May 1995

When discussing this topic with fellow students I remarked, that literate programming is not so widespread because of its young age. But my counterpart said that the X windows system was even younger and almost everybody is using it today. What do people think about that?

Apples and oranges. A fairer comparison might be OOP vs. literate programming. I think a big part of the "problem" of literate programming is that you can't hack in it. Hackery in the literate programming motif results in something far worse than the cruftiest perl script. Working with literate programming requires advance planning, and more than a little writing. Many programmers seem to avoid comments like the plague (dramatic case in point: Look at the source code to dvips some time. I suspect that this might be entirely comment-free).

From: **Will Ware**
Date: 25 May 1995

Norman Ramsey writes: ...the payoff for writing literate programs seldom justifies the expense. ...We have found that the cost of polishing a literate program to a ``publishable'' standard ranges from 1 to 4 times the original development cost... Unless a program is to be published as a book or a paper, making it literate is a waste of resources.

I'm currently writing a bunch of what might be considered semi-literate code. It's not meant to be publishable, it's just meant to provide a much higher level of documentation than most of the code I've seen (or written) around my company in the ten years I've worked there. At this level, literate programming seems to be a win, or at least an intelligent-looking gamble.

The marketplace rewards time to market. For new-product development, literate programming is a way to lose in the marketplace---the company that ships first wins, even when their product is full of bugs.

I guess I'm in a lucky position; I seem to be ahead of the hardware guys for the time being. I'm using the relative luxury now to literatize my code as fully as I can, so when the crunch inevitably comes, I'll have that as a foundation to work from. Documenting in the crunch tends to be pretty spotty, but my hope is to minimize the amount of code for which that applies.

As for the original question (Why is literate programming nearly unused?) my theory is that it's just a matter of cultural inertia. Most programmers I know haven't heard of it. As another post points out, literate programming is historically tightly linked to TeX, and tends to be restricted to systems on which TeX is available. In industry, a lot of software development is done on PCs running DOS and Windows, a world where TeX is rare, so the world of PC programmers is almost uniformly ignorant of literate programming.

I have also observed in some programmers, including at times myself, a certain queasiness about reordering source code. When something breaks, one is sorely tempted to study the tangled source code rather than the original document. The order of presentation in the literate programming document seems at those times to be artificial and perhaps deceptive.

I once discussed literate programming with a project manager. He understood the ideas of literate programming, but he was concerned that some programmers would take it as encouragement to pontificate beyond the point of diminishing returns. "How much is enough" is likely to be a highly subjective matter. Also, there may be a lot of programmers who are comfortable writing code but don't like the idea of their English writing skills being put on display, beyond the usual telegraphic comments.

From: **Stephen Mc Kearney**
Date: 25 May 1995

Norman Ramsey writes: My colleagues and I tend to use literate-programming tools, but not to polish our programs (as readers of the noweb source know to their cost). We have found that the cost of polishing a literate program to a ``publishable'' standard ranges from 1 to 4 times the original development cost. The polishing process improves the code as well as the documentation, but the process is usually too expensive to justify. As researchers, we get paid to have new ideas and develop new technologies (and to publish about them), not to write beautiful programs. Unless a program is to be published as a book or a paper, making it literate is a waste of resources.

Is literate programming not a 'model of research'? It probably depends on your area but in computing is publishing algorithms/programs not desirable. It is usual to separate the algorithm from the program and publish the algorithm but this is one of the criticisms of the Artificial Intelligence research --- publishing the imagined algorithm rather than the actual algorithm. There are some nice critiques of the AI program AM which discuss this issue. Unlike a sorting algorithm, the size of the program makes it difficult to extract the algorithm. TeX is presumably another example. Is this not one of the roles of literate programming? I have not had a chance to read it but I understand there is a C compiler available as a literate program. What about an electronic journal dedicated to 'research through programming'? This says nothing about industrial programming which Norman also mentioned. It is a totally different area with other objectives.

From: **Richard O'Keefe**
Date: 25 May 1995

Kristopher Johnson writes: 1) Lack of literate programming support in programming tools (editors, debuggers, profilers, etc.)

I'm currently reading "The Stanford GraphBase". Now Knuth is one of the all-time greats in computing. I 'cut my teeth' on The Art of Computer Programming. I read everything of Knuth's I come across. But having read about 1/4 of The Stanford GraphBase I am disappointed. (1) There are several places where a diagram would really have helped things. I know how hard it is to produce good-looking diagrams, but still... (2) Only 1/4 of it read, and I have found 9 problems with the C code, two of them show-stopping and quite unnecessary violations of the standard (the book is dated 1993; the C standard is 1989). I'm afraid code that takes for granted that calloc() will set pointers to NULL makes me feel ill no matter how beautifully it is typeset.(3) No amount of cute typesetting can substitute for a decent module system. I am aware of the cross-references, and they help a lot, but they are not intended to be a module system.

I don't care about debuggers very much; in my experience they are invariably a seductive waste of time. But if your literate programming tool emits the #line directive as it should, any good UNIX C compiler+debugger should cope. Has anyone tried CWEB with SPARCworks yet? Does it have any problem with literate programming? What I care about is static analysis tools. I not only can run lint on the output of an literate programming tool, I have done. Found some mistakes, too.

From: **pettengill**
Date: 25 May 1995

Richard O'Keefe writes: (2) Only 1/4 of it read, and I have found 9 problems with the C code, two of them show-stopping and quite unnecessary violations of the standard (the book is dated 1993; the C standard is 1989). I'm afraid code that takes for granted that calloc() will set pointers to NULL makes me feel ill no matter how beautifully it is typeset.

If you've read the interview with him on the Web, you know that he has a narrow focus which is limited to his current writing. He's found that he has to do this to have any hope of meeting his goal of completing the Art... by 2000 something. This does mean that he doesn't have time to follow the progress of the ANSI C standard. In fact, he admitted last summer that he wasn't even aware that there was an ANSI standard for C until this was pointed out to him, probably by people reading/user CWEB and/or Graphbase. And he's also admitted that he never considered that possibility that, 1) a NULL pointer wasn't equal to zero, and 2) physical memory order and pointer comparison order weren't equivalent.

Now consider this: If Knuth had not used literate programming, would anyone have noticed and would the code have failed unpredictably? Do you have an equal amount of code that you believe would have a lower defect rate if reviewed by 1000s of people? Are there software systems that control your life that you feel should receive even more scrutiny than Graphbase? You must admit that its rather an extrordinary compliment that you expect Don Knuth to know everything about any and every area of computing.

From: **John Daschbach**
Date: 25 May 1995

Norman Ramsey writes: Leaving aside from the usual NIH and similar problems, but I think there's a more fundamental problem: the payoff for writing literate programs seldom justifies the expense.

My colleagues and I tend to use literate-programming tools, but not to polish our programs (as readers of the noweb source know to their cost). We have found that the cost of polishing a literate program to a ``publishable'' standard ranges from 1 to 4 times the original development cost. The polishing process improves the code as well as the documentation, but the process is usually too expensive to justify. As researchers, we get paid to have new ideas and develop new technologies (and to publish about them), not to write beautiful programs. Unless a program is to be published as a book or a paper, making it literate is a waste of resources.

I have colleagues who write electronic structure code for massively parallel machines. This is code that will be around for a while and people take a long time getting up to speed with it. It would be hard to imagine a better place for literate programming, but they don't use it. There is still legacy code around in electronic structure that people can't maintain but it still works and the effort to change it is large enough to just use what works.

I think the problems are the lack of standard tools and a reluctance of people with years of Fortran experience to lose the familiar structure. Chunks are nice when you are reading the big picture, but when you have to chase down bugs or optimize having it all in standard notation is more comfortable. A tool that would produce well structured non-literate programming code from the literate programming code might help.

That said, it would be hard to find an application that would be better for literate programming than this type of work, but it's not being used. This is code that will almost certainly be around for many years, and need to be maintained as new theory and computational algorithms get added and as parallel machines evolve. If it's not being used for projects like this there seems little hope.

In a more production environment, things are no better---the marketplace does not reward quality in software (just consider the offerings of those who have made billions of dollars in software). The marketplace rewards time to market. For new-product development, literate programming is a way to lose in the marketplace---the company that ships first wins, even when their product is full of bugs.

So this is why Windoze95, still not shipping, and clearly inferior to OS/2 (out for years considering v2.0) is apparently winning the market.

The one hope in all this is that literate programming could dramatically reduce maintenance costs and increase the useful lifetimes of products. There are a few companies who have experimented with literate programming and gotten these kinds of results. Unfortunately, much of today's software is built with a ``get it out now'' mentality, not a ``build it to last'' mentality.

I often wonder what Knuth thinks of the state of things. He gave us TeX, but people for the most part don't want it. They want Word with all it's problems. He gave us Web, but people want Visual Basic. If people bought cars like software we would be happy with Gremlins.

From: **Jacob Nielsen**
Date: 25 May 1995

Ron Peierls writes: While all the barriers to widespread adoption of literate programming which have been suggested in this thread probably are significant, it seems to me that the major barrier is the lack of a single standard. As a result it is difficult, if not impossible, to share code even with other literate programming enthusiasts if they don't use the same flavor as you. If there were a standard even if it were at a low level with filters into and out of it for each WEB, then I think our crusades might be more successful.

The adoption of a Common Web Format (CWF) that all webs could be transformed into, would be better. This way one would only write two tools for each different WEB; one for the translation to CWF and one for the translation from CWF. Also, the CWF wouldn't have to conform to the WEB that does the least. A command that is not understood or implemented in the target WEB would just be lost.

With most WEBs a converted document would still produce the same source(s) and with a little extra work even the indexing of identifiers could be preserved [1]. The total number of commands in the CWF would be around the number of commands in ``WEB'' (or ``CWEB''), since they do the most... Anyone know af any caveats to this approach?

What about the use of different text divisions (CWEB: TeX and _one_ scrap per text part; nuweb: LaTeX and multiple scraps per text part (chapter, section etc)) ? I know that ``noweb'' can do conditional extraction[2] and that some WEBs (FWEB?) can use macros in scrap names and that would be impossible to use in another WEB.

[1]: Switching between noweb/nuweb and CWEB could be tricky, since noweb and nuweb uses user-defined indexes and CWEB can generate the indexes automatically. I suppose that the preservation of indexes requires that the WEBs uses similar approaches.

```
[2]: In noweb, you can say:

<<Do something ((UNIX))>>=
...

<<Do something ((DOS))>>=
...
```

and you can then choose which version of the ``Do something'' scrap you want to output. This way you can support different systems without using things like C's '#ifdef ... #endif' --- some languages can't do this kind of thing.

From: **Mary Bos**
Date: 25 May 1995

Norman Ramsey writes: My colleagues and I tend to use literate-programming tools, but not to polish our programs (as readers of the noweb source know to their cost). We have found that the cost of polishing a literate program to a ``publishable'' standard ranges from 1 to 4 times the original development cost. The polishing process improves the code as well as the documentation, but the process is usually too expensive to justify. As researchers, we get paid to have new ideas and develop new technologies (and to publish about them), not to write beautiful programs. Unless a program is to be published as a book or a paper, making it literate is a waste of resources.

I've happily worked as a maintenance programmer for over 20 years (in engineering, business, and MIS) and have seen plenty of "one-shot" programs that turned into corporate mainstays. I've seen some fairly "unmaintainable" code in my working career. So if even these "one-shot" programs had some literating (not necessarily to publishable standards), this would help me when I'm wading into the quagmire. Unfortunately, most maintenance programming is done by new programmers or new hires that aren't familiar with the code base, application, or company operations. These new people are not connected to the oral history or may not have programming maturity to understand certain items need to happen a particular order. This means the maintenance generally doesn't follow the original implementor's or designer's structure.

Another problem with maintenance programming is introducing the "software brittleness" or "software cancer" syndromes. Again if the literate programming was set up, we (the maintenance types) will not have the excuse of introducing the "brittleness" or "cancer" because we are given some more background to make the code better fit the original design (or lack of it). The real goal of maintenance programming is to make a program less error prone and less brittle with each maintenance incident, so why shouldn't the original (and maintenance) designers and implementors want to insure the intent in understood?

I think the value of literate programming as pointed out in other postings, is the process of making the reader sit back for a moment and take a breath before modifying the code. I know from my personal experience, it's locate the place for the enhancement or find the offending code, look to see what may be affected directly by modifying the offending code, modify, test, take a breath, and on to the next problem. But the full impact of the change is not always characterized, hence the one line changes that can cost a company several billon dollars because the code and variables immediately affected accepted the change but there is some very indirect variable or code chunk that is affected by the change. Usually this indirect code is trying to "kludge" a fix because the root cause was not easily located but the fix was simplier (or within the time limit given or ability of the person assigned).

From: **Bart Childs**
Date: 27 May 1995

Jacob posed a question ``Anyone know af any caveats to this approach? '' I know one, the more features you add to an interface the harder you make it to use. Web systems have frequently been criticised as being too hard to learn and too many things to learn. I agree that it is a load, but I think it is worthwhile to learn the full ones like: WEB, CWEB, and FWEB. I like my printed code to be pretty, a good index (automagically and liberally supplemented), keywords emphasised, ...

I have not really been joining into all these discussions lately, but I find most of the pleas for additional features and other NIH things to have simple work arounds. Incidentally, I think of the NIH in another way. I call it the syndrome of: ``God should have chose me to write Genesis'' Then, it dawns on me that one of these additional features in a recent laundry list would really help me. I write tests (yeah like for freshman problem solving courses) in fweb. It sure would be nice to have an escape back into TeX mode to create proper space, tables, lines, ... for answers to be put in. (If someone has a good work-around for that one, I would appreciate hearing of it.)

I see some merit to the Common Web Format and filters to convert them back and forth. However, I think extensive use of the ones that are out there and real sharing of the experiences is the right way to go.

A student of mine just finished her PhD and the data she took was based on the use of (the original) WEB, emacs, and web-mode in a CS/1 type course. Guess what? Freshmen Aggies (tamu stands for Texas A&M University and the A is Agricultural) learn all that with no problem. Well, there were some problems. Those with the most background for the course wanted to use the Turbo Environment and NOT LEARN anything else. Deborah Dunn and I will have a paper at the TeX Users Group conference in July. The paper should be finished in about two weeks and we will be pleased to make it available to those interested.

The strongest part turned out to be that lab projects were handed in twice weekly. First, the `outline' with some of the pseudo-code, maybe. Then, with that corrected and fleshed out. It is a real winning concept. I think the real answer is to get people to `think in terms of design, pseudo-code, and interative improvement.' Web (and web-mode) are naturals. (There is another real answer, don't let the students know there are a bunch of nay-sayers out there who think it is impossible, too much work, doesn't look like my grandfather would have written it, etc.)

From: **Jacob Nielsen**
Date: 28 May 1995

Ron Peierls writes: it seems to me that the major barrier [to the widespread use of literate programming] is the lack of a single standard. [...] If there were a standard even if it were at a low level with filters into and out of it for each ?WEB, then I think our crusades might be more successful.

Jacob Nielsen writes: The adoption of a Common Web Format (CWF) that all webs could be transformed into, would be better. This way one would only write two tools for each different WEB; one for the translation to CWF and one for the translation from CWF. [...] Anyone know af any caveats to this approach?

Bart Childs writes: I know one, the more features you add to an interface the harder you make it to use. Web systems have frequently been criticised as being too hard to learn and too many things to learn. I agree that it is a load, but I think it is worthwhile to learn the full ones like: WEB, CWEB, and FWEB. I like my printed code to be pretty, a good index (automagically and liberally supplemented), keywords emphasised, ...

We seem to want the same thing: everyone using the same (excellent) tool or at least being able to share the source. Even though the race seems to be closing, this is just not the case at the moment (I see people using CWEB, FWEB, noweb, nuweb and not much more) Although I can read different WEB languages quite easily, reading just isn't the same as being able to reuse the code easily.

Bart Childs writes: I see some merit to the Common Web Format and filters to convert them back and forth. However, I think extensive use of the ones that are out there and real sharing of the experiences is the right way to go.

As I see it, the problem isn't as much not understanding the CWEB/noweb et al. sources but rather the inherent trouble in reusing a piece of, say, CWEB source in a noweb source. With a Common Web Format I could transform The Stanford GraphBase into a noweb document and integrate the bits and pieces that I needed directly into my noweb WEB. But then again, the number of literate sources are not exactly plentyfull. Looking forward to the continued sharing of experiences and code.

PS: I'm not suggesting that we all should use a ``full'' WEB tool but the ability to exchange WEBs is important and as such the Common Web Format could be the way forward. The idea of "cut'n'paste" between CWEB, noweb, nuweb warms my heart.

From: **Norman Ramsey**
Date: 01 Jun 1995

Jacob Nielsen writes: The adoption of a Common Web Format (CWF) that all webs could be transformed into, would be better. This way one would only write two tools for each different WEB; one for the translation to CWF and one for the translation from CWF. Also, the CWF wouldn't have to conform to the WEB that does the least. A command that is not understood or implemented in the target WEB would just be lost.

I have my doubts about this approach because of the likely complexity of the CWF.

The total number of commands in the CWF would be around the number of commands in ``WEB'' (or ``CWEB''), since they do the most... Anyone know af any caveats to this approach?

Yes. CWEB & friends are too damn complicated, and unless the CWF is designed very carefully and tested thoroughly, it is going to be very hard to write tools for it. You will have to define very carefully what sorts of parts of the CWF different tools are required to understand.

You probably know that the implementation of noweb uses a similar approach, translating the source code into an intermediate form, which is then gradually transformed by a pipeline of individual tools. This is costly because Unix processes aren't free, but it means the individual tools can be small and simple. Most important, users can write tools that add new features without having to touch one line of noweb distributed code, and without haveing to recompile. For example the conditional-extraction feature you mention is implemented in a short tool written by Lee Wittenberg.

...noweb and nuweb uses user-defined indexes and CWEB can generate the indexes automatically.

Actually, noweb can generate indexes automatically for C, Icon, TeX, Standard ML, Yacc, and Object-Oriented Turing. Each of these index generators is implemented as another small tool. Except for C, which is notoriously hard to parse, these things are implmeneted in <30 lines of code each.

BTW, noweb also comes with a nuweb parser, so any nuweb program is automatically a noweb program (with slight loss of information). So you can use noweb to automatically add an index to your nuweb source. (The automatic indexes aren't as good as what programmers write by hand, because programmers are better at figuring out which identifiers should be indexed.)

So, without much evidence to justify it, I think the goal of automatic or semi-automatic translation from one source form to another is interesting. I think the goal of building ``universal'' tools based on a universal intermediate form is misguided---many literate-programming tools are too featureful to be duplicated easily, and I suspect you'll just end up creating yet another literate-programming tool :-(

From: **Lee Wittenberg**
Date: 03 Jun 1995

Norman Ramsey writes: I think the goal of building ``universal'' tools based on a universal intermediate form is misguided---many literate-programming tools are too featureful to be duplicated easily, and I suspect you'll just end up creating yet another literate-programming tool :-(

I agree. The best place to look for evidence is the realm of programming languages. How many of these were designed as "universal tools" to replace all previous languages? [I don't know, but I suspect that I can't count that high.] How many of these "universal tools" succeeded in actually displacing the languages they were intended to supersede? [zero]. As P.J. Plauger puts it ("Programming on Purpose II", Prentice-Hall, 1993): "We techies like to believe that the technical issues are most important.... So much of our future productivity (and fun) depends on the elegance of the languages we use. We'd better get it as right as possible. That viewpoint ignores an important truth, however. Each of us has his own notion of what constitutes the best technical solution. Put 50 techies in a room with a 300-page document and you'll get an amazing spectrum of opinions. Opinions about issues on every page. Just try to resolve each difference with a rational discussion aimed at achieving unanimity. I assure you the process will never converge. The simple fact is, we techies are human. (Perhaps some of your coworkers are suspect, but I don't want to hear about it.) We have emotional attachments to certain ideas that are impossible to factor out. As I've often remarked, the fewer facts we can marshal in defense of a technical opinion, the more zealous we become in arguing that opinion."

I think we will have different literate programming tools for quite a long time. I tend to prefer noweb (for its simplicity and extensibility), but there are times when using it that I really miss the features of CWEB. FWEB has (or had, I haven't looked at it in a long time) an interesting feature in that it automatically subscripts identifiers with the number of the chunk in which it is defined. I found this feature very disconcerting, but I can see how it could be useful, and how someone could get attached to it. We have to make tradeoffs in the tools we use, and each of us has (and will continue to have) a different set of criteria and priorities. Vive la difference! [Apologies for my -- possibly -- misspelled French.]

From: **Jacob Nielsen**
Date: 03 Jun 1995

Norman Ramsey writes: Yes. CWEB & friends are too damn complicated, and unless the CWF is designed very carefully and tested thoroughly, it is going to be very hard to write tools for it. You will have to define very carefully what sorts of parts of the CWF different tools are required to understand.

First let me explain that the reason for coining the Common Web Format was that it's less work to write 2*n than n*n translators if you want to exchange and reuse web code across different literate programming tools. That said, I do see problems with D.E. Knuth's tools in relation to other tools, and it's the not the prettyprinting.

The major problem with CWEB is that it limits you to make only one code chunk per section which almost no other tools does. D.E. Knuth has decided on the entire enviroment where many other tools lets you decide on how to divide your web (i.e. leaves the documentation parts alone.)

I believe the Common Web Format should support CWEB and friends but to deal with them properly, the CWF should not only deal with the code chunks but also with the documentation parts. This would render the whole idea of a CWF somewhat useless. The solution is to simplify the CWF (called ACWF -- Almost Common W... :-) and only let it deal with the chunks and the odd piece of code in the documentation. I suppose this will translate all webs into nowebs :-)

So, without much evidence to justify it, I think the goal of automatic or semi-automatic translation from one source form to another is interesting. I think the goal of building ``universal'' tools based on a universal intermediate form is misguided---many literate-programming tools are too featureful to be duplicated easily, and I suspect you'll just end up creating yet another literate-programming tool :-(

If one needs to exchange webs, the (A)CWF could be useful (2*n versus n*n translators) but I agree that the literate programming tools has to be similar in nature for this to work well. PS: I'm glad to hear that noweb can parse a nuweb web now.

Web for lex/yacc

From: **Don Hosek**
Date: 19 May 1995

Has anyone extended any c-web to allow for inclusion of lex and yacc code? It'd be really handy to be able to include the code for a lexer and parser in the same source file as the supporting c code.

From: **Norman Ramsey**
Date: 21 May 1995

noweb, nuweb, FunnelWeb, and FWEB all have this ability. Some are more recommended than others :-) If you want this indexed automatically, you're stuck with noweb. Try

```
noweave -autodefs c -autodefs yacc
```

If you want is prettyprinted, and you want the lex and yacc prettyprinted in a way different from the C, you're out of luck. If you want everything prettyprinted in C style (reasonable), you could try Kostas Oikonomou's prettyprinters with noweb.

From: **Don Hosek**
Date: 22 May 1995

Norman Ramsey writes: noweb, nuweb, FunnelWeb, and FWEB all have this ability. Some are more recommended than others :-) If you want this indexed automatically, you're stuck with noweb. Try

```
noweave -autodefs c -autodefs yacc
```

If you want is prettyprinted, and you want the lex and yacc prettyprinted in a way different from the C, you're out of luck. If you want everything prettyprinted in C style (reasonable), you could try Kostas Oikonomou's prettyprinters with noweb.

Now forgive me if things have changed in the meantime, but... I dismissed noweb out of hand when the original TUGboat article was

published for one simple reason: It doesn't allow me to have the code presented in a different order than the compiler wants to see it.

In actuality, as I've thought about the problem, what I'd really like to be able to do is to have a multilingual web. I've not thought too much about semantics yet, but here's a first thought off the top of my head. Including the language directive at the start of a file or module name makes that the language for the whole file or module. One can also have inlined code in other languages (e.g., inlined SQL or assembly code).

```
@ This section will generate the yacc file
@(@%yacc@>foo.y@>=
etc.

@ This section will generate the lex file
@(@%lex@>foo.l@>=
%{
    @%c@>
    @<Inlined C code for lex definition section@>
    @)
%}
@<Lex definitions@>
%%
@<Lex patterns@>
%%
@%c@>
@<Support code for the lexer@>
@)

@ Language modes should be inherited. This will automatically be in
lex mode
@<Lex patterns@>=
...

@ This will automatically be in C mode
@<Inlined C code for lex...@>=
...
```

From: **Sven Utcke**
Date: 22 May 1995

Don Hosek writes: I dismissed noweb out of hand when the original TUGboat article was published for one simple reason: It doesn't allow me to have the code presented in a different order than the compiler wants to see it.

Are you sure you aren't confusing this with cnoweb? cnoweb (by, if I remember correctly, the same author) is nothing more than a clever hack to enable you to use TeX-documentation within your C-program. Not what one would really call literate programming. noweb, on the other hand, is a fullgrown literate programming tool quite similar to the one proposed in your article, and does indeed allow for the reordering of code. It is well worth trying (although I use FWEB 1.40, which can do what you want but, to my knowledge, is still beta).

From: **Norman Ramsey**
Date: 23 May 1995

Don Hosek writes: I dismissed noweb out of hand when the original TUGboat article was published for one simple reason: It doesn't allow me to have the code presented in a different order than the compiler wants to see it.

ALERT! noweb != cnoweb

You have confused noweb with another tool having a similar name. noweb has never been written up in TUGboat. The only article written about it appeared in the November 1994 IEEE Software. noweb meets all of Thimbleby's criteria for literate programming, including: flexible order of elaboration (code in any order), ``verisimilitude'' (derive code and documentation from one source), and automatic indexing and table of contents. Automatic indexing is available for a limited number of languages; for other languages noweb supports Preston Briggs's ``semi-automatic'' indexing (which produces better results when applied with fanatical discipline).

In actuality, as I've thought about the problem, what I'd really like to be able to do is to have a multilingual web... Including the language directive at the start of a file or module name makes that the language for the whole file or module.

Others have suggested labelling only the ``root chunk'' and having that carry forward to the whole refinement. Whether the labelling should be embedded in or external to the source file is a subject of debate.

Sven Utcke writes: Are you sure you aren't confusing [noweb] with cnoweb? cnoweb (by, if I remember correctly, the same author)

noweb and cnoweb are unrelated tools with different authors.

SPIDER generated WEB for Lisp?

From: **Paolo Amoroso**
Date: 22 May 1995

I was wondering whether anyone generated - or is planning to work on - a WEB system for Lisp/Scheme with SPIDER. If none exists, what's the most similar tool? (I know about SchemeWEB, but it seems too simple to me)? I'm interested in true literate programming systems, not in pretty-printers or formatters like the excellent SLaTeX.

From: **John Ramsdell**
Date: 23 May 1995

I once started to construct a Lisp/Scheme WEB with SPIDER only to find that SPIDER made assumptions about identifiers than are not true of Lisp identifiers. For example, if I remember correctly, hyphen is not allowed in identifiers. This is part of the reason I ended up creating SchemeWEB, a tool you accurately call simple.

From: **Lee Wittenberg**
Date: 23 May 1995

I have a prototype Spidery Scheme grammar, if you're interested. accepts standard S-expressions for code, but typesets them as McCarthy's original M-expressions, which are much more readable by humans, IMO. You can write me direct if you'd like me to email you a copy. It's very primitive -- I only did enough to prove to myself that such a thing was possible, but it's got enough to get you started on a more complete one. BTW, noweb, nuweb, and the other language-independents are perfectly fine for webbing Lisp & Scheme, if you like reading S-expressions (which I don't :-).

From: **osmanb**
Date: 28 May 1995

I think there is scheme.spi. I tried it once and it worked fine. Check the CTAN.

Perfect literate programming tool gripe

From: **Dietrich Kappe**
Date: 24 May 1995

Something that all of our favorite literate programming tools are weak on is in providing support for alternate typesetting languages. You can do LaTeX or TeX, but HTML and RTF are only provided as an afterthought through *TeX2* filters. Here are the various literate programming tools that I have used and their limitations as regards alternate typesetters:

Funnelweb: Has a typesetter independent input language, but the back end is tightly wed to TeX. LaTeX2HTML support provided through an independent patch.

Noweb: Unix bound. Most of the tools assume TeX or LaTeX as the output language.

Nuweb: Output and scrap indexing tightly wed to LaTeX.

CWEB : *TeX all the way through.

My question: Is there any way we can get some sort of standard into the literate programming tool universe? Maybe an intermediate

language or a common library for typesetting code. These standards need not focus around a least common denominator. Please post, if only to tell me I'm all wet.

From: **Norman Ramsey**
Date: 25 May 1995

*Dietrich Kappe writes: Something that all of our favorite literate programming tools are weak on is in providing support for alternate typesetting languages. You can do LaTeX or TeX, but HTML and RTF are only provided as an afterthought through *TeX2* filters. Please post, if only to tell me I'm all wet.*

You're all wet. noweb and nuweb support HTML directly, without any TeX involvement. The next release of noweb (to be made today, I hope) will also support flat ASCII, thinks to Phil Bewig.

Noweb: Unix bound. Most of the tools assume TeX or LaTeX as the output language.

Noweb runs quite comfortably on DOS, thanks to the porting of Lee Wittenberg and Phillip Miller. Noweb supports TeX, LaTeX, or HTML. I'm not aware of any noweb tool that assumes TeX or LaTeX. What made you think otherwise?

From: **Mark Kramer**
Date: 31 May 1995

Here are the various literate programming tools that I have used and their limitations as regards alternate typesetters:

Funnelweb: [...]

Noweb: [...]

Nuweb: [...]

CWEB : [...]

CLiP: totally independent of programming language as well as formatter or word processor. Well, to be honest: the only restriction is that the word processor should have an ASCII export.

My question: Is there any way we can get some sort of standard into the literate programming tool universe? Maybe an intermediate language or a common library for typesetting code. These standards need not focus around a least common denominator.

This is something different. Moreover, IMHO it's contrary to your point above: then you are not hooked up to (La)TeX, but to the new intermediate language.

From: **Will Ware**
Date: 31 May 1995

Dietrich Kappe writes: Is there any way we can get some sort of standard into the literate programming tool universe? Maybe an intermediate language or a common library for typesetting code. These standards need not focus around a least common denominator.

Interesting idea. I have started reading up on noweb, which is implemented as a chain of operations on a sequence of files. The first step in the chain is to translate the source document into an interlingua, designed for easy parsing in awk/Perl/Icon/C, and subsequent processing is done on this interlingua, for both tangling and weaving. The inner workings of noweb are described in Norman Ramsey's article, "Literate Programming Simplified", in the Sept 94 issue of IEEE Software. Noweb's interlingua might make a good starting point for a universal interlingua, since its current role within noweb is already pretty close to this.

From: **Pete Johnson**
Date: 11 Sep 1995

Don Grodecki writes: I have the notion that these days HTML might be a great system for doing literate prigramming, instead of TeX. One advantage would be the ability to present literate programs easily on the WWW. Does anyone have any thoughts on this? Has anyone ever seen anything like this?

There are a number of systems which will output HTML, I am sure many people here can point those out. I do some literate programming using SGML as the markup language rather than inventing my own markup. This has the advantage of a standard markup language with off-the-shelf tools to support it. Tools include parsers, formatters, and editors. In addition to being programming language independant, it is also formatter independant. My approach is to modify a formatting DTD (in this case HTML) to include literate programming constructs. These, as a minimum, define code chunks and refer to them. Here is an example:

```
<code id="Initialize variables">
i = 0;

<code id="Process stuff">
<cref id="Initialize variables">
i++;
```

I usually then have a script to expand the code tags and generate the program, as well as a script to expand the code into markup (in this case HTML). Doing things this way seems to be a good choice if you want to output code marked up as SGML (HTML falls into this category), if you like the idea of using some standard for markup (after all, literate programming is markup of program source code), and you can stand the SGML syntax (which I agree is unasthetic, but not too bad). Let me know if you have more questions, and I will try to answer them.

From: **Colman Reilly**
Date: 11 Sep 1995

Don Grodecki writes: I have the notion that these days HTML might be a great system for doing literate prigramming, instead of TeX. One advantage would be the ability to present literate programs easily on the WWW. Does anyone have any thoughts on this? Has anyone ever seen anything like this?

Obviously it can be a useful thing to do. Noweb already supports it. Actually, at the moment I'm working on a developers editor that would allow WYSISLWYG[1] editing of documents and code and would export to HTML or TeX. It's in Tk4.0, so it should port to Windows and Mac as soon as the Tk port for them comes out, which is supposed to be RSN. It should also support lots of other things needed for project development, like RCS and probably storage over the Web. It uses noweb as a backend, since I see no reason to duplicate Norman's excellent work. Prototype coming soon, since all I'm doing is using Tcl to glue things together.

[1] What You See Is Sorta Like What You Get, which is as good as you can do with TeX or HTML. It won't support tables or formulae initially, or any of TeX's complicated features. It *will* let you embed raw TeX and HTML though.

From: **Sven Utcke**
Date: 09 Jun 1995

I wonder if anyone out there could help me with one or both of the two FWEB-problems (I'm using FWEB 1.40 (October 30, 1993)):

1) FWEB and HTML. What is the way to go if one needs both an online (html?) as well as a printed version, and if both versions will need to contain images and formulas? LaTeX2html won't do (at least not Version 0.7a4 (Wed Dec 21 1994)). Will I have to get xhdvi? I really would prefer to be able to use netscape (however, if this isn't possible, so be it). Could some kind soul outline the way to go?

2) verbatim Language (@Lv) in FWEB. While most of my current work is written in C, some of it needs to be done in a verbatim kind of language where formating is important (and FWEB shouldn't add any additional information, please). However, I can't get this to work at all. In the best cases, formatting just is destroyed (lines can easily exceed 72 characters). In the worst cases, ftangle breaks. Will this change with a newer (1.51?) version? Note, however, that I can't use LaTeX2e for several reasons. Any help would really be appreciated

---

From: **Norman Ramsey**
Date: 29 Jun 1995

Sven Utcke writes: What is the way to go if one needs both an online (html?) as well as a printed version, and if both versions will need to contain images and formulas?

I know of no literate-programming tool that solves this problem. HTML is a disaster for images and formulas, and all the answers are hacks.

LaTeX2html won't do.

Indeed not. I bet it's going to require some serious work on someone's part before progress is made here. Suggest you ask

comp.text.tex and comp.text.sgml.

2) verbatim Language (@Lv) in FWEB. While most of my current work is written in C, some of it needs to be done in a verbatim kind of language where formating is important (and FWEB shouldn't add any additional information, please). However, I can't get this to work at all.

noweb and nuweb handle verbatim text just fine. With noweb, you could use the Oikonomou prettyprinters to format the C code. Mixing C and verbatim in one file would require some adapation, but it's an afternoon hack...

The development cycle and literate programming

From: **Paolo Amoroso**
Date: 25 May 1995

During software development, the output of the specification and design stages is often a series of documents containing text, diagrams, tables, formulas, etc. Since a literate programming system: supports the implementation stage, is able to integrate all the above mentioned documents, doesn't usually force a particular order of creation and allows their production by successive refinement, Is it wise to use such a system also in the first stages of development? In other words, how early in the program development cycle can literate programming tools be effectively employed? Any opinion - or account of direct experience - will be highly appreciated.

From: **Lee Wittenberg**
Date: 25 May 1995

I think so -- right from the start. In fact, I think literate programming can be a logical extension of what Bertrand Meyer is talking about when he advocates using a programming language as a specification and design language as well. Once the specifications are set, code chunks can be added as necessary in the appropriate parts of the document. The code chunks will be close to the specification of the feature they implement, and therefore, can be more easily checked.

When I do literate programming, I frequently leave chunks open for future development. In fact, one of the projects I'm working on is a design for a prototype system, in which the purpose of the code chunks is simply to illustrate roughly how things can be put together; I don't intend to actually tangle and compile the web. On the other hand, if the design seems workable, the web might very well form the basis for the actual prototype.

One of the advantages I've found with literate programming is that *It Doesn't Have To Be Publication Quality* (take note, Norman :-). What's important is that the thoughts get down on paper. Incompletely expressed thoughts are better than no thoughts at all. In my experience, it's much easier to check text chunks for accuracy than traditonal comments, partly because of the different typography, but more, I suspect, because text chunks use complete sentences, while traditional comments usually don't. The discipline of writing a complete sentence tends to promote accuracy a bit more than that of writing sentence fragements. It may be, of course, simply that literate programmers are just better communicators than anyone else. :-)

Even "semi-literate" programs, like the noweb source have advantages over traditional programs. *The Chunk Names Themselves Carry Information.* In particular, they show the programmer's thought processes with respect to design and stepwise refinement. Even though the noweb code is mostly unadorned code chunks (incompletely indexed), I have no compunction about browsing around the code, looking for information. The chunk name <<make all those damn active characters ``other''>> is a particular favorite of mine. My nocond macros make use of information gleaned this way to typeset conditional chunks interestingly, and to change the "(not defined)" and "Root chunk" messages to "(conditional)" and "Conditional chunk", respectively. It was significantly easier to find what I needed than it would have been in a straight-line LaTeX style file.

From: **Patrick McPhee**
Date: 26 May 1995

Paolo Amoroso writes: [...] In other words, how early in the program development cycle can literate programming tools be effectively employed?

Wayne Sewell argued in favour of using literate programming tools to write specifications (in "Weaving a Program").

From: **Don Wallace**
Date: 05 Jun 1995

I am fascinated by the title of this forum... Can anyone tell me, if practical, what the formal definition of literate programming is? I assume that literate programming has to do with the writing of source code, and alludes to construction of programs that are humanly readable as logical descriptions of processes. (True?) Does it also encompass such practices as commenting of code?

FWIW, I have been witness over the years to what one could only call 'illterate programming'... cryptic variable names, non commented code, inexplicable spaghetti, all ignorantly defended by the idiot-savant programmer/propellor head/whining prima donna-with-an-attitude of the moment as being 'necessary' because 'we need to get this out.'

If the above hypothesis of what literate programming is, is correct, then I am (sadly) afraid that 95% of the slop that is written in the commercial world is truly <<illiterate>>... In fact most programmers I have known probably do not read more than the the back of the cereal box in the morning... while admirable, literate programming is swimming against the tide.

Web systems and TeXinfo?

From: **Mike Eggleston**
Date: 28 May 1995

I like the idea of literate programming and I have been using the concept for some time by actually writing my database designs in TeXinfo and then writing a perl script that analyzes the database design and creates the database schema from the document.

What I'm asking is, is there a more formal or an already existing method of doing this? I like TeXinfo for being able to use one document that is both hardcopy and on-line and it is also an easily analyzed format for my database systems.

I haven't seen any other mechanism that has both a hard-copy and an on-line format for the document. My systems need to be bound into a manual and also available on-line for the help-desk/support personnel to trouble-shoot and help users and correct problems when an application or a batch procedure doesn't run properly.

Well, thanks for the read and I'll be looking forward to any responses and ideas. I think posting to the group would be best... maybe there are other people lurking here that are interested in the same things. How about extending the TeXinfo macros and the makeinfo, etc., programs to enable TeXinfo to be used as a literate programming system? TeXinfo documents can be printed or used as a hyper-text type system already, it just needs a way to do what the tangle and weave programs do. Any thoughts?

From: **Will Partain**
Date: 29 May 1995

Mike Eggleston writes: How about extending the TeXinfo macros and the makeinfo, etc., programs to enable TeXinfo to be used as a literate programming system?

The Glasgow literate-programming stuff is sorta like this. The markup language *looks* like LaTeX, but it's really glorified Texinfo. From there, you do lit2pgm (tangle), or lit2{texi,latex,html}. We don't make a lot of noise about the system, but it works reasonably. Available from ftp.dcs.glasgow.ac.uk, in pub/haskell/glasgow/lit2x-0.19* (including a README). (NB: I am not sure that the lit2html support is too good in that version...) The system is a big mass of perl gunk; not too bad to hack on.

From: **Lee Thomas**
Date: 31 May 1995

Mike Eggleston writes: I like the idea of literate programming and I have been using the concept for some time by actually writing my database designs in TeXinfo and then writing a perl script that analyzes the database design and creates the database schema from the document.

I'd really be interested to see the Perl script.

Actually, one of my interests is literate programming in scripted languages like Perl, TCL/Tk/Expect, etc., and SQL. I have a tool available to me that will strip Unix "man" pages out of source code (more-or-less "inverting" the literate programming paradigm of stripping code from what started as a document) by searching for tokens. Of course, it's inadequate for my purposes, which are to fully explain my scripts to a naive maintainer, not to merely give a manual page which tells how to run it. Imagine, if you will, 2000 lines of finely-tuned and moderately-commented modular Perl scripts dropped in your lap.

In general, I believe literate programming has limited real-world acceptance partly because programmers are tending toward interpreted languages, and the available tools (with the glaring exception of something like a TeXinfo system) are targeted more toward a particular compiled language. Many problems are caused by a script which started as a quick hack and grew into a critical

system, without comment.

I haven't seen any other mechanism that has both a hard-copy and an on-line format for the document. My systems need to be bound into a manual and also available on-line for the help-desk/support personnel to trouble-shoot and help users and correct problems when an application or a batch procedure doesn't run properly.

Another limitation of literate programming is the target of TeX/LaTeX as the document preparer of choice, because 1) most places I've worked have either no TeX system administrator, or a disinterested/lackadaisical one, so the system is not maintained - plus, it's multiple steps to print the document; 2) once it's printed, a document is out-of-date, or at least suspicious; and 3) there are N printed copies and N+1 interested parties, leading to "borrowed" documents which are not returned, which leads to multiple copies with different revision dates, ...

Putting the documentation online solves much of this. I've tried to use tools like dvips and ghostview, but they're usually poor substitutes for actual paper copy, alas. TeXinfo looks like a reasonable choice. I'm glad to see noweb adding an HTML target, because people here are putting their documents on the World Wide Web - even material that is viewed only within a small group of engineers is most easily browsed via Mosaic or Netscape.

I think posting to the group would be best... maybe there are other people lurking here that are interested in the same things.

Yup.

From: **Norman Ramsey**
Date: 01 Jun 1995

Lee Thomas writes: Actually, one of my interests is Literate Programming in scripted languages like Perl, TCL/Tk/Expect, etc., and SQL.

Yes! This was one of the goals motivating the design of noweb back in 1989. In fact, one of the coolest things you can do is this:

```
#!/bin/sh

PATH=...

<<EOF notangle | perl
(literate program here)
EOF
```

So that you only have the literate source file and you create the script on the fly (not for the faint of heart when there are errors).

Another limitation of literate programming is the target of TeX/LaTeX as the document preparer of choice, because 1) most places I've worked have either no TeX system administrator, or a disinterested/lackadaisical one, so the system is not maintained - plus, it's multiple steps to print the document; 2) once it's printed, a document is out-of-date, or at least suspicious; and 3) there are N printed copies and N+1 interested parties, leading to "borrowed" documents which are not returned, which leads to multiple copies with different revision dates, ...

HTML might be a better solution, since almost everybody has a WEB browser these days, and it can easily be turned into PostScript or ASCII. noweb and nuweb support HTML.

Putting the documentation online solves much of this. I've tried to use tools like dvips and ghostview, but they're usually poor substitutes for actual paper copy, alas.

I think so, too.

I'm glad to see noweb adding an HTML target, because people here are putting their documents on the World Wide Web - even material that is viewed only within a small group of engineers is most easily browsed via Mosaic or Netscape.

I am really hoping to get funding for adding automatic cross-file indexing support to noweb. I have the plan; I just need the money...

From: **Marc van Leeuwen**
Date: 06 Jun 1995

Lee Thomas writes: Putting the documentation online solves much of this. I've tried to use tools like dvips and ghostview, but they're usually poor substitutes for actual paper copy, alas.

Is it just sloppiness that you forgot to mention any plain dvi-previewers here or did you actually not try any of these? If you want to view TeX-output on the screen, there is really no point in converting to postscript; in fact TeX files viewed via ghostview usually look horrible, because the (bitmapped) fonts have been included at a resolution unfit for displaying on the screen. On the other hand a previewer like xdvi can select bitmaps according to the screen resolution, and if your screen has grayscales (or colours) it can use greyscale anti-aliasing to improve the apparent resolution even beyond the actual resolution of the screen; all in all the quality is incomparably better then via dvips|ghostview. Since xdvi also automatically rereads dvi-files when they are changed, it allows you to view TeX output almost immediately as it is produced; moreover it can even display included postscript images (albeit not as fast as the surrounding text). The hypertext additions of xhdvi should give it all the possibilities you could wish for a browser of TeX output, but unfortunately the version of xhdvi I have seen is still buggy. If you are not using X-windows, there must be other other applicable vi-previewers with similar properties.

From: **Lee Thomas**
Date: 06 Jun 1995

Marc van Leeuwen writes: Is it just sloppiness that you forgot to mention any plain dvi-previewers here or did you actually not try any of these?

Neither sloppiness nor ignorance, Marc - but thanks for at least giving me *that* much choice. It's been 5-7 years since I last worked at a site which had xdvi, and which had decent sysadmin support for TeX/LaTeX. (Some days I wish Knuth would charge $5,000 per copy of TeX so I could convince management that it's superior to the other - commercial - systems.) When I last tried it, it worked well. However, the dvi still needed to be processed into PostScript to be printed (I couldn't avoid printing), and we encountered differences between the online and the printed versions. Trying to get it all to work together became a time sink I couldn't afford. I very much want a system which allows me to programmatically produce text that looks identical when rendered on-screen or on-paper.

If I could be the TeX administrator everywhere I work (and job changes are a fact of life), I would install xdvi in a heartbeat. However (trust me), altogether it's just not workable in most of the U.S. high-tech environments I encounter. If I need to abandon LaTeX (temporarily?) so I can get literate programming happening where I work, so be it. Prioritizing is painful, but often necessary.

From: **Lee Thomas**
Date: 08 Jun 1995

Mike Eggleston writes: I haven't seen any other mechanism [besides texinfo] that has both a hard-copy and an on-line format for the document.

HTML, which is supported by nuweb and noweb. In my own work I do best by using a simple subset of latex that is easily converted to HTML. That way I get page numbers on the hard copy, but I don't clutter up the online version with meaningless numbers. Of course, I have to resist the temptation to do things in latex that can't be done in HTML...

Encapsulation in literate programming

From: **Kristopher Johnson**
Date: 17 Jun 1995

I'm writing a fairly large program in C++, using CWEB for the first time. I find myself doing stuff like this a lot:

```
@ We need a class to represent something.
@<Class declarations>@ +=
    class Foo {
    public: @<Foo public member declarations@>
    private: int privateMemberDataValue;
    };
```

```
 ... (dozens of pages later, far away from the discussion of class Foo)

@ In function f(), we need to access a Foo instance's data. We'll
  add a new accessor function to the Foo class.
@<Foo public member declarations@> +=
   int GetDataValue() { return privateMemberDataValue; }
```

In short, instead of declaring a class's data and functions all in one place, I add stuff where I need it. I wonder whether this practice is good or bad. Here are my thoughts:

It's bad: the author is obviously making this stuff up as s/he goes along, with no forethought or design. A reader cannot possibly understand the essential nature of a class if it does not have a simple, concise declaration, all in one place, and no one can easily make changes to the classes involved. It's the literate programming equivalent of spaghetti code.

It's good: this is how the process of designing classes really works. A class designer starts with a vague idea of which classes are needed, and then refines the design iteratively. The CWEB file chronicles the iterative process, and can explain why certain features have been added to a class. Thus, although classes are intermingled throughout the document, the associations between them are made clear. The indexing and cross-referencing provided by CWEB make it easy for a reader to find all declarations for a class, if that is desired.

It depends: the goal of an author is to make the reader understand, so authors must use their own judgments and refrain from relying on simple rules.

I welcome comments.

From: **Jacob Nielsen**
Date: 18 Jun 1995

Or: The literate programming paradigm is just a tool; don't expect it to do everything. There's no substitute for thinking a little ahead. Literate programming is just so nice, because: 1) You can keep all the documentation (diagrams, analysis, code) in the same place and intermix them as needed. 2) We can present the actual code in pieces.

There's nothing wrong with a text only introduction or even actually writing the important methods down from the start, but leave the implementation out of it---this also means *not* listing ``private'' data before they are actually needed. [Yes, I never place code in the C++ class interface; I use 'inline' instead.]

Object Oriented Analysis and Design is an iterative process and not introducing the private data before they're actually used, eases the iteration process. It's good if you take the time to explain the overall purpose of the class(es). The description should, IMO, go into a seperate ``chapter''. I've found that keeping the relevant data together with the methods that alter them is *good*; it makes altering so much easier.

From: **Bill Rassieur**
Date: 18 Jun 1995

IMHO: it is best to treat a class as a type, and a "something" for which we desire a nice, easy to grasp, model for. Therefore, I say that the introducing methods with that kind of commentary you describe is "bad." (well, it's not evil, just hard to read...) I have found that when I write comments like:

```
@ Okay, in this function we need to access a data member, so let's add
it above.

<>=
...blah blah...
```

that it *does* serve me, the author, but it does not serve the reader. I think that I should delete comments like this on a second pass over the Web. Imagine reading a mystery novel where Hercule Poirot (or choose your favorite character) discovers some clue, and then you read, "and it was okay that he found the gun under the bed because earlier, the bad guy secretly put it there, and the bad guy had time to do it because..."

So what should the reader be presented? I think that at the class declaration, all of the methods and data members should be described. Hopefully the class makes enough sense as an abstraction that each of the member functions' operations relates in a more or less common sense way to the class. If it is a "spacecraft" class and it has a "fuelReading" function, that's fine, but a "cleanUpFiles" function might not be clear. You the author are free to provide whatever interface that class may require (i.e., you may not have been Godlike in anticipating all members when you first penned the declaration, and you are allowed to make changes before you release the program to the public). The reader gets a polished version and sees the class documentation. In those 12-pages away locations where you reference a member function you are free to write some commentary that connects up the current use of a member with the class. That's the style that I as a reader would prefer (my taste). As an author I *want* the kind of commentary you first mentioned, but I promise (on good days) to polish before others read it.

From: **Marc van Leeuwen**
Date: 19 Jun 1995

To me, the only criterion for deciding to order to present a literate program is what is the best way to explain the program to someone not familiar with it (like the next person to maintain the program, or the friends you give the program to, or yourself in a year's time). This may take a significant amount of planning, or rearranging the code if you are not satisfied with your original setup (which is fortunately relatively easy using literate programming), but it is certainly worth the effort for any serious program. This criterion could favour either a localised or a distributed presentation, depending on the specifics of the program at hand.

From: **Bart Childs**
Date: 19 Jun 1995

This makes at least the fourth response to this question. I agree with the earlier answers and each of us have addressed a slightly different part of the answer. As in the original post, it depends upon the view as to which way it is best? The view of the maintenance programmer or the author?

I wish to add that if it is written for the author, then the author should seriously consider the maintenance programmer as well. That does not necessarily mean that it should be reorganized as soon as it is working. (I assume that) CWEB does the same as the original. That is, the section numbers for a class are differentiated. The number for a definition of a class gets underlined. I think the Knuthian index is one of the best things about literate programming.

Further, for a Knuthian or Knuth-like index, the author would add a number of user-supplied index entries. In a number of cases, it is advantageous to reference section numbers of the class definition. I suppose that some systems may not allow forward reference, but backward reference should always be done. The need for smaller scope of appearance of variables, classes, ... can be reduced by good indexes, but not totally eliminated. Often people dismiss literate programming without making total use of the facilities and I think this is a good example of how the whole should be used.

From: **Lee Wittenberg**
Date: 20 Jun 1995

I find your "bad" arguments interesting, but I think the "good" arguments are really what literate programming is all about. A web doesn't necessarily represent the order of design (although I've had a great deal of success with "stream of consciousness" webs) -- see Bart Childs's recent post -- I think the technique you describe is easier for a reader to understand the various pieces of the design. It's not necessary, for example, to understand anything about the class constructors when you're interested in a specific member function. Your technique allows the writer (and, therefore, the reader) to concentrate on a small piece of the class at a time. IMO, this is the greatest strength of literate programming.

From: **John Haxby**
Date: 21 Jun 1995

I think its half way. If the specification of the class is spread all over the place then potential users of that class have a hard time in finding out what the class actually does. Similarly, if the representation (client instance data) is spread around all over the place, then implementors of the class have a hard time in finding out what data is stored and what the representation invariant is.

I'd say that the external specification (public interface) needs to be kept together in one place and the internal specification (representation, representation invariant and private utility routine specifications) need to be kept in another. Everything else can, and probably should, be somewhere else -- that way, users and implementors of the class (extensions) get to find out what the class is, exactly, without being distracted by unnecessary detail.

Of course, you're in trouble if you change the representation invariant to any significant degreee since you have to find the implementation of all the functions in the class and make sure that they work with and adhere to the new invariant, but that's nothing to do with literate programming, that's to do with getting the design more-or-less right to begin with. And luck. Interestingly, both CLU and Modula3 (to differing extents and Modula2 as well) have a tendancy to enforce this kind of approach anyway. The trouble with C++ is that you can't live with it, or live without it. :-)

From: **Kristopher Johnson**
Date: 22 Jun 1995

My original example was probably a little too extreme. A better description of what I'm doing is this:

1) Declare the "core" classes first, along with their primary data members and functions. By core classes, I mean those that represent the things that the application deals with; for example, in a word processing app, the core classes would be Paragraph, Style, Font, and so forth.

2) Declare and define I/O functions and any necessary related data members for all the classes. Since all the classes use the iostreams library, putting all the I/O stuff together makes sense.

3) Declare and define the classes for windows, panes, menu bars, etc., that will make up the user interface. Add any functions and members to the core classes that are necessary for communication/integration with the user interface classes.

4) ... and so on ...

So, although declarations and definitions for the core classes are scattered throughout the program, all the I/O stuff is together, all the user interface stuff is together, etc. I like this approach. The core classes may not be well encapsulated, but similar functions are put together, so I don't think the reader will get lost. I think the organization by feature/functionality, rather than by class, actually helps.Comments?

Why doesn't CWEB have true comments?

From: **John Bay**
Date: 19 Jun 1995

Why doesn't CWEB have true comments? The manual (for version 3.0) says of "@q" that "The control text that follows will be totally ignored--it's a comment for readers of the CWEB file only." But that's not precisely true because the context of the "@q" control text is restricted, so the text can't contain other control sequences, for instance. The motivation for this is that I would like to use Sccs revision control with CWEB source files. But Sccs introduces control sequences of its own - things like "@(#)" that cause CWEB to complain.

From: **Marc van Leeuwen**
Date: 20 Jun 1995

While it is true that the control text following `@q' is totally ignored, it is not true that it can consist of an arbitrary sequence of characters (the control text has the same specification as control texts used for other purposes). Note that the same is true for any comment in any language, since there must always be a way to terminate a comment. On the other hand you are right that "@q" is more restrictive than comments usually are: any undoubled `@' sign either signals the end of the control text or an error. Guessing reasons why this was done, the following come to mind: (1) it is simpler to use the same lexical rules for similar constructs (control texts); (2) having only `@>' recognised after `@q' might lead to a quiet loss of control codes if `@>' is forgotten or mistyped (this is not so likely because contol texts are limited to a single line anyway).

The code `@q' is a fairly recent addition to CWEB; the traditional way to keep something out of print is to use TeX comments (after `%'). However, like in contol texts, `@' must be doubled if it is used in a TeX comment, so this does not solve you problem either. I believe that `@q' was only added because you cannot use TeX comments in C mode; for instance in

```
@ Some commentary.
@< A module @>= { some_code; }

@i auxiliary.file

@ A later section.
```

the beginning of `auxiliary.file' belongs to the C-code portion of the previous section, and therefore one cannot use `%' there. (This convinces me once more that it would be a good thing to have and ``end of section'' code that does not immediately start a new section; why is this idea not used in any literate programming system?).

As for `Sccs', you can only make this work if Sccs would accept `@@(#)' in place of `@(#)'. However, I feel that this whole problem is due more to Sccs than to CWEB: apparently it assumes that any language has sufficiantly flexible commenting conventions to encapsulate Sccs's junk strings without causing any harm. There are more tools that make such assumptions, but there really is no justification for them, and CWEB will certainly not be the only language causing problems. The proper solution to the problem is that if tools like Sccs insist on placing certain strings in source files, then they should also provide tools to strip these strings before actual processing of those files.

From: **Lee Wittenberg**
Date: 20 Jun 1995

Personaly, I prefer to have my version control info displayed in the woven web (I use RCS rather than SCCS, though). I use several macros that typeset the important parts and ignore the strange stuff. For the Log, though, which I don't want to appear, I simpy use the TeX comment character % in a text chunk. So you can always use % in a text chunk as if it were a "true" CWEB comment. E.g.,

```
@*Index.
%
% $Log: cwebatq.html,v $
% Revision 1.1  2001-04-07 15:54:19-07  daniel_mall
% <>
%
% Revision 1.0  2000-11-16 07:16:26-08  daniel_mall
% Initial revision
%
%
% ... expanded stuff from $Log: cwebatq.html,v $
% ... expanded stuff from Revision 1.1  2001-04-07 15:54:19-07  daniel_mall
% ... expanded stuff from <>
% ... expanded stuff from
% ... expanded stuff from Revision 1.0  2000-11-16 07:16:26-08  daniel_mall
% ... expanded stuff from Initial revision
% ... expanded stuff from ...
```

From: **Stephen Boyan**
Date: 21 Jun 1995

in all languages comments are restricted in what they can contain because of the need for comment-terminators

In record-oriented languages like COBOL and old FORTRAN dialects, which were designed for the IBM mainframe, a record (or punched card) oriented machine, comments were indicated by placing a special code in a special column (say column 7). The comment could then contain anything EDCDIC could encode. This positional, or semi-positional, character of these languages makes parsing them interesting.

Literate programs for Microsoft Windows applications

From: **Steve Furlong**
Date: 20 Jul 1995

I'm pretty new to literate programming, but I'm already enthusiastic about it. In the abstract, that is. Much of my work is on Microsoft Windows, using a variety of application generators and coding environments. None of the literate programming tools I've examined, and that includes almost all listed in the FAQ, handle many issues involved in this situation.

1. For the most part, Windows programs require a handful of source files even for "hello world". These include the .c and .h files, plus .def, .rc and often others. What is the best way to keep all of these files in a literate system? Have .w files for the .c and .h and let the (usually smaller) files just reside as themselves, or have .w files for every source file, even if the source file is only five lines long?

2. Many Windows programs these days are created with application environments, app wizards, and so on. Examples include the Microsoft, Borland, and Symantec C++ development environments, Visual Basic, and Delphi. Source code is (or can be) ASCII text, which could be post-processed into a literate file, except that the various environments usually want to read the source files time and again, and wouldn't be able to handle the strange files.

One (poor) solution is to create the first pass of the app, then remove it from the environment, turn it into a literate program, and never put it into the env again. This of course loses most of the benefits of the integrated environment. We could also use very large comments to capture the design decisions, but that isn't a literate program. More importantly, it has been pretty well demonstrated that most people don't do it.

Another option is to include specially-formatted comments to serve as links to sections in a separate text file. Winword's macro language is powerful enough to merge the doc and the source files. The problem is that this means maintaining at least two files, in separate environments. I can't see this working in practice on anything larger than a toy "proof of concept" project.

The only workable solution I see to the problem of incorporating maximum decision and rationale info into the source files in an integrated environment is to modify the environment itself to allow tying a word processor to the source code windows. This is more than a simple customization to any env I've worked with, which is most.

Unfortunately, the apps developed in this situation are among those most needed to capture decision info. Too many Visual Basic programmers, for instance, just hack together an app any old way, kludging around language limitations with no thought for

maintainability.Ideas, anyone? Borland, are you listening?

From: **Lewis Perin**
Date: 22 Jul 1995

Steve Furlong writes: 2. Many Windows programs these days are created with application environments, app wizards, and so on. Examples include the Microsoft, Borland, and Symantec C++ development environments, Visual Basic, and Delphi. Source code is (or can be) ASCII text, which could be post-processed into a literate file, except that the various environments usually want to read the source files time and again, and wouldn't be able to handle the strange files.

Not just that - the debugger may not know what to do with a webfile. This was a galling defect of Symantec 6.x, but Borland succeeds here. I haven't tried Symantec 7 on this issue.

One (poor) solution is to create the first pass of the app, then remove it from the environment, turn it into a literate program, and never put it into the env again. This of course loses most of the benefits of the integrated environment. We could also use very large comments to capture the design decisions, but that isn't a literate program. More importantly, it has been pretty well demonstrated that most people don't do it.

Another option is to include specially-formatted comments to serve as links to sections in a separate text file. Winword's macro language is powerful enough to merge the doc and the source files. The problem is that this means maintaining at least two files, in separate environments. I can't see this working in practice on anything larger than a toy "proof of concept" project.

The only workable solution I see to the problem of incorporating maximum decision and rationale info into the source files in an integrated environment is to modify the environment itself to allow tying a word processor to the source code windows. This is more than a simple customization to any env I've worked with, which is most.

Unfortunately, the apps developed in this situation are among those most needed to capture decision info. Too many Visual Basic programmers, for instance, just hack together an app any old way, kludging around language limitations with no thought for maintainability.Ideas, anyone? Borland, are you listening?

Sorry, but I haven't any encouragement to offer. In my experience, where a Windows-based application has a separable subsystem that has little if anything to do with the GUI it's quite possible to make it a literate programming subproject, but for the rest the advantages of using the application framework's facilities are just too great to forgo. Faced with this I just write profuse C++ comments and remind myself that I can still use Emacs and CWEB for other projects.

From: **Lee Wittenberg**
Date: 22 Jul 1995

Steve Furlong writes: I'm pretty new to literate programming, but I'm already enthusiastic about it. In the abstract, that is. Much of my work is on Microsoft Windows, using a variety of application generators and coding environments. None of the literate programming tools I've examined, and that includes almost all listed in the FAQ, handle many issues involved in this situation.

1. For the most part, Windows programs require a handful of source files even for "hello world". These include the .c and .h files, plus .def, .rc and often others. What is the best way to keep all of these files in a literate system? Have .w files for the .c and .h and let the (usually smaller) files just reside as themselves, or have .w files for every source file, even if the source file is only five lines long?

I do some Windows (literate) programming myself, so my experience may be helpful. I find that noweb is quite good for combining different source files in different languages. My hello.nw source would contain

```
<<hello.c>>=
<<hello.h>>=
<<hello.def>>=
<<hello.rc>>=
```

chunk definitions, and the makefile uses notangle's -R option to extract the appropriate files at the appropriate times. The only file I don't put into the web is the makefile itself, as it leads to an "interesting" chicken-and-egg problem.

2. Many Windows programs these days are created with application environments, app wizards, and so on. Examples include the Microsoft, Borland, and Symantec C++ development environments, Visual Basic, and Delphi. Source code is (or can be) ASCII text, which could be post-processed into a literate file, except that the various environments usually want to read the source files time and again, and wouldn't be able to handle the strange files.

One (poor) solution is to create the first pass of the app, then remove it from the environment, turn it into a literate program, and never

put it into the env again. This of course loses most of the benefits of the integrated environment. We could also use very large comments to capture the design decisions, but that isn't a literate program. More importantly, it has been pretty well demonstrated that most people don't do it.

I find that integrated environments impose too much of their own ideas of how I should work, so that I generally go back to the command-line compiler and a makefile, where I have complete control. The newer (v4.5) Borland C/C++ compilers have the capability to let you add your own tools, and specify how they are to be used in a project. I've gotten noweb working with them, and CWEB almost working (there's a bug in the 16-bit compiler that prevents it from compiling with the "official" PC change files. The 32-bit compiler works okay, but it's only for Windows, but Borland does not provide an EasyWin library for Win32, and I just haven't had the time -- or inclination -- to do anything about it), so that might be a way for you to go. [BTW, even with this feature, I still prefer the command line compiler.]

We could also use very large comments to capture the design decisions, but that isn't a literate program. More importantly, it has been pretty well demonstrated that most people don't do it.

That's somewhat the way CLiP works, so it *can* be literate programming.

From: **Jacob Nielsen**
Date: 23 Jul 1995

Steve Furlong writes: 1. For the most part, Windows programs require a handful of source files even for "hello world". These include the .c and .h files, plus .def, .rc and often others. What is the best way to keep all of these files in a literate system? Have .w files for the .c and .h and let the (usually smaller) files just reside as themselves, or have .w files for every source file, even if the source file is only five lines long?

Having a web file for each of the smaller files is not smart. IMHO, making a one to one maping of files are a bad thing, especially when those files are small. Making a one to many maping (one web file, multible output) is much better, especially when the output depends on each other (e.g. foo.c and foo.h usually depends on each other). The tools nuweb and noweb both support the one to many mapping and I'm sure many others does as well.

2. Many Windows programs these days are created with application environments, app wizards, and so on. Examples include the Microsoft, Borland, and Symantec C++ development environments, Visual Basic, and Delphi. Source code is (or can be) ASCII text, which could be post-processed into a literate file, except that the various environments usually want to read the source files time and again, and wouldn't be able to handle the strange files.

[begin Disclaimer] I've never used any of those application enviroments that spits out code you have to edit---reading Steve's article, I'm glad mine just spits out objects. [end Disclaimer] Giving the problems integrating literate programming and application enviroments under Windows, I guess the only way is to seperate the GUI and the program. This should enable you to write the computation part using literate programming and struggle with the GUI using two interlinked documents or just verbose commenting. Ah well, lets hope Borland sees the light soon...

From: **Paolo Ciccone**
Date: 25 Jul 1995

Jacob Nielsen writes: Ah well, lets hope Borland sees the light soon...

Well, I'm listening but I'm not the person that will set the decisions for BC++ or Delphi. I just finished porting FWEB to OS/2 (same executable run under DOS) and the whole literate programming stuff is very interesting, I'm experimenting with an internal tool, converting gradually the code from pure C++ to Web. Personally I think that it's not very difficult to integrate literate programming in the BC++ IDE but the automatic generated code is another story. Borland constantly reads the newsgroups for C++ and I know that recently other groups have been opened for Delphi. I suggest to post there to make the right people aware of your needs.

Using smaller margins in noweb?

From: **Robert McLay**
Date: 17 Aug 1995

I have just started using noweb. (version 2.7a). While I like a lot of the features, I don't see how to change the margins that noweb uses. It appears to me that it uses margins appropriate for a book. Is it possible to have the latex output have approximately 1 inch margins instead of the default 1.5 to 2 inch margins.

From: **Norman Ramsey**
Date: 17 Aug 1995

This is a common complaint about latex. It has nothing particularly to do with noweb. There may be a `fullpage.sty' file floating around (try the CTAN index) that creates smaller margins. Alternatively, you can grub around in \textwidth and all the other horrid page-layout parameters.

From: **Lee Wittenberg**
Date: 17 Aug 1995

noweb uses the default LaTeX margins for whichever style (or class) you are using. You can change them by changing the values of \oddsidemargin, \evensidemargin, and \textwidth in the preamble of your document. However, I don't advise doing it, particularly to have 1 inch margins.

The LaTeX margins are chosen to provide approx. 60 characters/line when typeset, which studies have shown is optimal for reading text. Any more than that causes severe drops in comprehension. 1 inch margins with a pica typewriter produce a 60-character line, which is why you were expecting them, even in a typeset document. If you really want thinner margins, use a larger font size (e.g., the 12pt option). LaTeX will generate smaller margins, but maintain the 60 chars/line.

Literate programming GUI?

From: **Anthony Yen**
Date: 20 Aug 1995

I've been giving some thought to a literate programming GUI, and would like to hear what others have come up with. I'll start off by saying that I believe that Knuth's "expository mode" should be expanded. An literate programming environment should allow groupware-like processes, with annotations, hypertext links, views (show/hide various levels of commenting), version control, and feedback from the entire spectrum of the development process (design, implementation, testing, user feedback). Two to three years ago I heard about an ambitious project underway at Chevron that had some of these points as its goals, but ever since then I haven't heard of anyone else trying to do the same thing. Comments?

From: **Kristopher Johnson**
Date: 20 Aug 1995

I think that OpenDoc support would be a great thing for the expository mode, allowing one to add pictures, sounds, movies, interactive demos, hypertext links, or whatever. I'm in the midst of designing an OpenDoc-savvy literate programming environment for the Macintosh. If anyone else is doing something similar, or has any comments or suggestions, I'd like to hear about them.

From: **Lee Wittenberg**
Date: 20 Aug 1995

I don't think that one tool should be expected to do everything. What I think we want is an environment that allows third-party tools (whatever they are) to work together easily. Unix pipelines allow this in a text-based environment, but most operating systems do not allow tools to work together easily. In addition, most commercial software environments ("groupware" and other) do not provide the necessary hooks for third party tools to work with them (and in the USA, it it not necessarily possible to figure out the hooks yourself legally -- in Europe it is legal to disassemble a program for the express purpose of providing an add-on; that's not necessarily true in this country).

I think the goals of the Oberon project (as I understand them) are applicable here. What we need is a user environment that is designed to be easily extensible, not just for literate programming, but for the tools to come that cannot yet be foreseen.

From: **Kevin Purcell**
Date: 21 Aug 1995

Kristopher Johnson writes: I'm in the midst of designing an OpenDoc-savvy literate programming environment for the Macintosh. If anyone else is doing something similar, or has any comments or suggestions, I'd like to hear about them.

Great -- I'd been thinking along the same lines for a literate programming environment for the Mac and OpenDoc seems to be an ideal technology to apply to this problem. If you'd like to discuss it off the list send me some private mail.

A rate limiting step to this work is the availiblity of OpenDoc development parts (i.e. project managers, compilers, linkers, "industrial strength" editors and so on) to support this kind of development. I don't want to give up all the advantages of GUI coding and debugging to do literate programming. And I don't want to have to provide all the parts either. I suspect Metrowerks will be thinking about this after CW 7.0. What work has already been done in this field? Can anyone point me to any interesting references (paper or on the 'net).

It seems most of the research has been done by people that use non-GUI environments under Unix so adding a few extra items to the make file is not a problem. Norman Ramsey's noweb also seems to be a step in the right direction to freeing literate programming from WEB's grip. One problem (as Lee Wittenberg alluded) is that current IDE's are really "too intergrated". On the Mac an IDE that was scriptable, attachable and tinkerable would provide and excellent foundation for developing these type of tools.

Literate programming in industry

From: **Jim Sisul**
Date: 21 Aug 1995

I've been studying and experimenting with literate programming for a few years, but have not seen it used in a commercial setting. Are there any studies out there that have examined the success of incorporating literate programming into a commercial software development environment? Do any of you use literate programming at work, or know of companies who do? Any pointers to studies of literate programming in industry would help me a lot. Thanks for your help.

From: **Norman Ramsey**
Date: 23 Aug 1995

Jim Sisul writes: Any pointers to studies of literate programming in industry would help me a lot.

Norman Ramsey and Carla Marceau. Literate programming on a team project. Software---Practice & Experience, 21(7):677-683, July 1991. Also, Joseph Brothers at Motorola has an unpublished MS that I expect will be released sometime.

From: **Lee Wittenberg**
Date: 24 Aug 1995

I don't know of any such studies, but I do know that Tipton Cole+Co.,in Austin, Texas, makes use of noweb "in a commercial setting". Carl Gregory (gregory@cs.utk.edu), who does not (I believe) subscribe to this group, can give you more information. Hope this helps.

From: **Anthony Yen**
Date: 27 Aug 1995

One aspect of literate programming which I have been unable to find references to is how literate programming is supposed to handle team-based development. In general, I have been quite dissatisfied with the lack of material discussing how groups can share information over distributed systems; there seems to be little that has been done on this subject in an empirical manner, as contrasted with a lot of the HI work that currently goes on with GUIs in the usability labs. Can someone point me in the right direction for my second try at a literature search? Thanks in advance.

From: **Mary Bos**
Date: 31 Aug 1995

Is anyone other trying to incorporate literate programming into their maintenance efforts? We are still circling around trying simultaneously sell the scoffers and getting our minds, fingers, and literate programming around over 250 KAELOC of code. Any strategies, suggestions, or things you found that did (or did not work)?

From: **Mary Bos**
Date: 06 Sep 1995

Some time ago, some one asked about who or where is literate programming used in industry. We are still trying to get literate programming introduced into our work group. We maintain several diverging code streams (each stream has about 260 KAELOC of c code, we also have assembly code, shell scripts, configuration files, release notes, and other bits of documentation to work with). This code is very intertwined, so splitting this code into "clean" partitions at first is not feasible (and still meet our deadlines). The base code has been in existence for over 10 years, hosted onto several platforms. Our code is used in real-time systems, so this 260 KAELOC of code ends up into one large executable file. This is not like some of the IS systems, where the system is 250 KAELOC, but there are individual executable files built to make the system.

Three of us, Joseph Brothers, Gary Young, and myself (Mary Bos) are trying to bend noweb into working on this large segment of code. We conservatively estimate of about 3 lines of documentation (including x-refs, indicies) per line of c code. We figured if we just put our code into noweb, we'd overrun the chunk indicies and main index as well as use about 50 reams+ of paper. Anybody have rule of thumb they use for estimating the documentation size? Especially the ratio between lines of moderately dense 'c' code to documentation?

I haven't seen any discussion of measuring the effectiveness of documenting the code (such as fog indicies). I know real software engineers are trying to make the process work and produce products on time, on budget, and on quality. But how will we know when we get there? Unfortunately managers like numbers (especially if the numbers look like productivity numbers). Is anybody trying to introduce literate programming into legacy code or has done so? I have been out exploring on the WWW and found software maintenance and legacy code have a dearth of citations.

From: **Adrian Zimmer**
Date: 03 Nov 1995

Here is a posting I've made to comp.software-eng and a couple other programming news groups. Readers of comp.programming.literate will find little new, but both the posting and the discussion it generates (if any) may be interesting. A use for literate programming. Literate programming was invented by Donald Knuth and used by him to develop his TeX system. Literate programming flows from the observations that software in textbooks is easier to understand than software in computers and that part of the reason is that software in textbooks is embedded within human readable explanations.

With literate programming, you write a textbook explanation of your code and include some annotations to identify the embedded code. Knuth built two tools to help. One of these, weave, creates cross references and formats your text for printing. The other, tangle, produces code for your compiler/interpretor. Several newer versions of these tools are now available. See "ftp://ftp.th-darmstadt.de/pub/programming/literate-programming/Tools/"

Writing a textbook, or even just a chapter of a textbook, to explain and present a computer program is quite a bit more work than merely writing the program. In the approximately twenty years since it was invented, literate programming has not become a significant tool in producing production software. For literate programming to be useful, the need for understanding has to be large enough to support the extra work of writing the program in a literate style.

I believe that scripts written for adaptation by others have this need. Almost every mass distributed program worth its salt now has a scripting language. It is not uncommon for a programmer to need to deal with several of scripting languages during a year. Many encounters are rather brief -- the programmer customizes a tool for a user and goes on to something else. Because of this, the programmer may not know the scripting language very well and may rely on example scripts to see what needs to be done.

Relying on example scripts can be frustrating and time consuming. Why was a script written the way it was? You want to know so you can figure out what changes are possible. However, explaining why is difficult. Sure the script presenter can write comments, but comments are a poor substitute for the flexibility literate programming gives you in your presentation style.

Here's an example of how explanations are presented in the literate style. This example shows a subroutine do do lexical analysis written in pseudocode that is based on the Perl scripting language. (Literate programming is NOT limited to Perl.)

```
sub advance {
 return unless $Toke;
 $Toke = '';
 local($Where,$OldCur);
 for(;;) {
   if( <<a new Toke is found in Buffer>> ) {
      <<put Buffer up to Toke into Prev>>
      $StartChunk = 0;
      return;
```

```
   } else {
      <<put Buffer into Prev>>
      if( $Final ) {
         return;
      } else {
         &newBuffer;
      }
   }
 }
}
```

Pseudocode statements appear this way

```
<< ... >>
```

The rest is Perl. When this literate program is tangled, pseudocode phrases are replaced with Perl code. When the literate program is woven for reading a WWW browser, each pseudocode phrase becomes a hyperlink to the code it represents. For example,

```
<<a new Toke is found in Buffer>>
```

is a hyperlink to one line of Perl code. Here is that line with a header that identifies the pseudocode phrase it defines.

```
<<a new Toke is found in Buffer>>=
($OldCur=$Cur) <= $LastCur && &search
```

The explanation for why this one line of code works is much longer than one line. If that explanation were placed as a comment it would make the code unreadable. If it were explained externally, the link between explanation and code would be tenuous. If it were missing, the side effects would drive you crazy. Literate programming strikes the right balance.

Some of you, of course, are saying "don't write code with such side effects". Here, it is true that one visible side effect could be avoided. However, in scripting languages, side effects are often the only way things get done. I'm giving an example of how to live with side effects.

If you are writing scripts for others to adapt, you can save time for those people by writing in a literate style. Even if they don't have a literate programming tool, they can read what you have written with a WWW browser. If they do have a tool, they will find adaptations of your script even easier. In short, literate programming adds value to your scripts.

Now, suppose you need to adapt a script for a spreadsheet. One spreadsheet publisher provides examples written in a literate style and the other does not. The example written in a literate style will save you time. This provides a competitive edge for the spreadsheet maker who writes literate examples.

Or, suppose you must get your example script from a third party. One source is free but not literate. The other costs a little but is literate. I believe many of us would pay a small fee to obtain a literate version. So, if you are writing scripts for others to adapt, then do look into the possiblity of writing them in a literate style.

By the way, if you are running Unix with Perl, Latex and a WWW browser, you may want to look at a beta version of a literate Perl script I am completing. It implements a little markup language that translates easily to Latex and HTML. A command line option makes documents written in the language usable with the noweb tool for literate programming. Contact me for more information.

CWEB and C++ templates

From: **Stefan Thienel**
Date: 23 Aug 1995

Who knows a trick to bring cweave/tex to typeset the function bad() containing the template allocation as nicely as the function good(). Here the line break after bad() is missing, but an undesired line break after good is inserted. Omitting the format definition "@s ARRAY int" causes cweave/tex to set everything in one line. I would like to have ARRAY typeset in bold face.

```
@
@s ARRAY int
@c
  void good()
  {
    int *vec;
```

```
  vec = new int[10];
}

void bad()
{
  ARRAY<double> *array;
  array = new ARRAY<double>(10);
}
```

From: **Lee Wittenberg**
Date: 23 Aug 1995

Try "array = @[new ARRAY<double>(10)@];" or various other placements of @[and @]. The @[@] pairs help to keep the parser from getting too confused. They are particularly invaluable in C++, where the syntax is often ambiguous at the level investigated by the CWEAVE parser.

From: **Stefan Thienel**
Date: 01 Sep 1995

Some people encouraged me to post a solution: Lee Wittenberg gave me a helpful hint and after some experiments I found out

```
array = new @[@,ARRAY<double>(10)@];
```

produces a good result. The control codes @[@] cancels the undesired line breaks. Using @, gives some extra space in the output such that it looks even nicer. This solution is not very elegant (who knows a better one?) but it works.

Literate programming without TeX

From: **Keith Ballinger**
Date: 24 Aug 1995

OKay, I have been reading this list for a few months now, and have read Knuth's stuff on literate programming. I am really excited by it all, and have been incorporating it into my own programs to an extent. However, I haven't been using any tools, such as FWEB or noweb, but instead just using the paradigms.

Now it seems to me that one of the major problems at this point with literate programming tools is the emphasis on using TeX to format. I have nothing against TeX, but I just don't find a big use for it. I don't like to print out my stuff, but instead prefer to view documents online.

However, I do think that the ability to write the program in both the bottom-up and top-down approaches that literate programming tools allow us are very nice. Matter of fact, I think this is the truly revolutionary aspect of literate programming: That I can ignore compiler standards of where I have to put different things, and can instead develop my program in my own fashion.

So this is what I would like to do. I would like to write a program that will take an ascii text file that is the *already formatted* document. (Using whatever word porcesor I want and saving it in ascii form.) This document is usually the end product of the "weave" command on many webs, but instead of weaving the document, I just will create it with my word processor. The program will basically tangle out the code into a compiler ready source code file. So basically I am asking for input on this. What do you all think? Has this some major drawback that I am missing? Would it be worth my time? Any input you have would be greatly appreciated.

From: **Lee Wittenberg**
Date: 25 Aug 1995

Keith Ballinger writes: Now it seems to me that one of the major problems at this point with literate programming tools is the emphasis on using TeX to format. I have nothing against TeX, but I just don't find a big use for it. I don't like to print out my stuff, but instead prefer to view documents online.

Different people work in different ways. That's why there are so many programming languages (and tools). I prefer reading paper, myself.

However, I do think that the ability to write the program in both the bottom-up and top-down approaches that literate programming tools allow us are very nice. Matter of fact, I think this is the truly revolutionary aspect of literate programming: That I can ignore

compiler standards of where I have to put different things, and can instead develop my program in my own fashion.

I agree completely.

*So this is what I would like to do. I would like to write a program that will take an ascii text file that is the *already formatted* document. (Using whatever word porcesor I want and saving it in ascii form.) This document is usually the end product of the "weave" command on many webs, but instead of weaving the document, I just will create it with my word processor. The program will basically tangle out the code into a compiler ready source code file. So basically I am asking for input on this. What do you all think? Has this some major drawback that I am missing? Would it be worth my time? Any input you have would be greatly appreciated.*

Take a look at CLiP and WinWordWEB. They work the way you describe. There is also a Framemaker-based tool, but I'm afraid I can't recall its name (sorry).

C(++) to WEB translation?

From: **Gilbert van den Dobbelsteen**
Date: 04 Sep 1995

I am new to this newsgroup and I have a question which is probably simple to answer. Is it possible to code in C(++) and after the coding process is complete, converting the created code to WEB. I know c2cweb does this but I want something more complicated. Demonstration follows below:

```
/* This is a simple C++ program that should be translated into a
   multiple section web.
*/

//@<The main stuff>=
main()
{
     //@<Print'm>=
     printf("Hello world\n");
     //@End
}
//@End
```

This should be translated into a web and result into 2 sections: The main stuff and Print'm. Is there a tools which can do this, or must I write one of my own?

From: **Lee Wittenberg**
Date: 04 Sep 1995

I think you'll have to write your own tool, but if the comments you indicated are already in the code, a fairly straightforward Awk script should do the job.

From: **Marc van Leeuwen**
Date: 05 Sep 1995

I don't know of the existence of any such tool, but then I also don't understand why exactly you would wish to have it. If you intend that the `//@' comments are already added in the coding process, then I think this is more tedious than writing the thing directly in the WEB format; if on the other hand you have to insert them after the coding stage, then again this seems more work than reshaping the code directly into WEB format by hand (I have a lot of practice with this first stage of LP-ifying plain code, and with an editor with good cut-paste, move and change-indentation facilities, it is an easy, albeit somewhat boring chore). The advantage of WEB format that makes it preferable in both these cases is that by moving the sub-module out of the way, it is much easier to percieve the structure of the surrounding code. (For this reason you are sometimes even forced to work bottom-up in locating candidates for sub-modules, even when top-down would make it much easier to find the right descriptions for the sub-modules; this is simply because the global structure cannot be discerned initially.)

Whatever approach you take, it is of course important to realise that isolating sub-modules is only a first step towards making a literate program from an illiterate one; it is vital that commentary is added to the sub-modules (except in some cases where an elaborate

module name already says it all), so that the understanding of the code you needed in order to break things up properly will not be lost before you come round to look at the code a next time.

CWEB cross references to sections

From: **Kendall Shaw**
Date: 05 Sep 1995

I'm sure these are FAQs but I've been wishing to see more literate programming content in this group, so I hope it is okay. 1. Can I make cross references to sections in the text, e.g.:

```
@*\label{token}The Gobbledy Gook section.
<@Lots of neat stuff@>=
1==1;

@*The Other Section.
This the other section (but you should really look at \ref{token}
where other things are discussed).
```

where \ref would resolve to the section number and name?

2. How do I get C code to be formatted more or less verbatim, as far as tabs and spaces. I could use a tabular environment in TeX, but I can't do that right, since it needs to be code for ctangle?

From: **Lee Wittenberg**
Date: 06 Sep 1995

Kendall Shaw asks: 1. Can I make cross references to sections in the text, e.g.:

```
@*\label{token}The Gobbledy Gook section.
<@Lots of neat stuff@>=
1==1;

@*The Other Section.
This the other section (but you should really look at \ref{token}
where other things are discussed).
```

where \ref would resolve to the section number and name?

In plain TeX CWEB, you can use \secno for backward references (this is simple):

```
@*The Gobbledy Gook section.
\edef\tokenlabel{\secno}
<@Lots of neat stuff@>=
1==1;

@*The Other Section.
This the other section (but you should really look at \tokenlabel
where other things are discussed).
```

For forward references, however, you'll have to "roll your own" LaTeX-like .aux file mechanism. CWEB.STY, on the other hand, probably allows you to use LaTeX \labels and \refs, just as in your example. I haven't tried it myself, so I couldn't say for sure.

2. How do I get C code to be formatted more or less verbatim, as far as tabs and spaces. I could use a tabular environment in TeX, but I can't do that right, since it needs to be code for ctangle?

You can use @= ... @> for small bits of verbatim code, redefining \vb if you don't like the frame, but if you want all the code set in a typewriter font exactly as you typed it, you probably want to use a tool like noweb, which doesn't do any prettyprinting. One of the main features of CWEB is its prettyprinting, so it's really a losing battle trying to turn it off. If that's what you want, you'll probably be happier with another tool.

Some questions on noweb

From: **Alberto Meroni**
Date: 29 Sep 1995

I have been till now an unliterate programmer and I will be very interested in turning to literate programming, but I experience the following problem. My program (fortran77) should usually be used by people which do not want to use this paradigm, so I would like to write them in literate style and then to have them assmbled in a conventional way so to be used and worked on by others, and this should include the comments and eventually the docs, turned in comment. I understand that in general this is not possible, has no one ever tried to do something like this (with eventually some restriction?).

From: **Stephen Boyan**
Date: 29 Sep 1995

How to write a literate program for the illiterate - ie, for those not using the paradigm, in this case with FORTRAN program maintenance.

noweb has a feature for doing this. It embeds the literate text as comments for, I believe, the following chunk. The feature is called nountangle. The DOS port I have for noweb, and which I haven't upgraded in over a year, does not contain nountangle, but I was told if I installed a UNIX emulator like the MKS Toolkit I could porbably run the UNIX version. Haven't tried this yet, but hope to do so for COBOL.

From: **Norman Ramsey**
Date: 03 Oct 1995

Roy Mathew wrote: 1) Is there a plan to allow for "parameterizable" chunks (i.e: define a chunk that takes arguments).

There has been a proposal from Lee Wittenberg on the table for over a year. I think it is a good proposal. The problem with it, and with all other such proposals, is that it requires language-dependent lexical analysis to decide what is and is not an instance of a formal parameter. Since half of you got on the net in the last year, I'll summarize.

```
Define
<<sum of [[x]] and [[y]]>>=
x+y
@ and you can now write:
<<*>>=
if <<sum of [[1]] and [[2]]>> ~= 3 then
  stop("something is very wrong")
@
```

2) Is there an emacs mode anywhere for noweb editing?

Thorsten Ohl has written a reasonably good one (I use it). There are some bugs. I don't know if it's available to the public :-)

3) Can you point me to some good examples of C++ code development under noweb.

Not I.

It seems that noweb offers significant advantages in the C realm, but many of those are rendered superfluous in C++ [ie: because C++ allows for in-place declarations, methods are name-scoped and therefore more-or-less self documenting...]

And exceptions, if they were real enough that you could count on them. Same thing happens in modula-3. On the other hand, noweb is *really* useful if you are programming in the original awk (the onewithout functions) :-)

From: **Robert McLay**

Date: 02 Nov 1995

I have started using noweb and I like it. The problem I have is with the automatic definition detection. For this dummy code below, I would hope that main, a, b, and c would be automatically detected but it doesn't for me. I did

```
noweave -autodefs c -index a.nw > a.tex
latex a.tex; latex a.tex; latex a.tex
```

and there are no indexed variables. Is this the way it is suppose to work? Or have I not compiled something correctly? Is this a bug or a feature? (;->)

```
@
\section{Introduction}

\bigskip
@
<<*>>=
main(int argc, char *argv[])
{
  <<main body>>
}

@
<<main body>>=
   int a,b;
   double c;

   a=0; b=1;
   c = (double)(a+b);
   printf("c: %f\n",c);

@
\section{List of code chucks}
\nowebchunks

\section{Index}
\nowebindex

\end{document}
```

From: **Norman Ramsey**
Date: 03 Nov 1995

Robert McLay writes: I have started using noweb and I like it. The problem I have is with the automatic definition detection [for C].

The bad news is that this detection is approximate and heuristic. The good news is that is is only a hundred or so lines of Icon. I would like to experiment with using the lcc front end to provide completely accurate cross-reference, but right now I have neither time nor funding for such an experiment.

```
main(int argc, char *argv[])
```

Maybe didn't get indexed because you didn't declare a return type, or maybe because you didn't put { on the same line.

```
<<main body>>=
   int a,b;
   double c;
```

Didn't get indexed because if they don't start in column 1, they must not be ``important'' enough to index. You can obviously change the autodefs code to suit your own style and prejudices...

Program prettyprinting macros

From: **Carlos Felippa**
Date: 29 Sep 1995

I am curious as to whether "prettyprinting" macros for TeX-inline listing of C++ and Fortran90 program source have been developed. More specifically, at some point in a TeX document I would like to say

```
\prettylistingCpp { programfilename codesection}

\prettylistingFortran { programfilename codesection}
```

where the codesection is delimited by keywords in programfilename. The program section listing then replaces the invokation. At the present I use simple verbatim-listing macros.

By "prettyprinting" I mean Weave-like quality. But implementation through Weave-as-filter has three serious problems: 1. It would require process spawn and synchronization, which I dont think TeX can do. 2. Weave carries a lot of extra baggage when the only objective is prettyprinting standard source code, rather than a Web source. In fact I don't know if Weave can process a file that does not have any Web commands. 3. Weave is not generally available with commercial implementations of TeX, such as TeXtures (which is the one I use most). Question: is a facility of this type, or some approximation to it, available in the archives?

From: **Jacob Nielsen**
Date: 03 Oct 1995

Carlos Felippa writes: I am curious as to whether "prettyprinting" macros for TeX-inline listing of C++ and Fortran90 program source have been developed. More specifically, at some point in a TeX document I would like to say

```
\prettylistingCpp { programfilename codesection}

\prettylistingFortran { programfilename codesection}
```

where the codesection is delimited by keywords in programfilename. The program section listing then replaces the invokation. At the present I use simple verbatim-listing macros. Question: is a facility of this type, or some approximation to it, available in the archives?

It seems to me that your problem is that your program (TeXtures) only invokes TeX and cannot invoke another program before/after TeX? If you can use emacs (on a UNIX system), then take a look at nuweb-mode, which invokes nuweb prior to TeX -- It works like a charm. I don't know of any TeX macros that handles prettyprinting by it self, so I guess the solution is to hack a literate programming tool to do what you want. Noweb supports prettyprinting of C (not C++!) and I believe that FWEB can prettyprint FORTRAN and C++ code. BTW, I think that the TeX teletype is quite nice in any case :-)

From: **Marc van Leeuwen**
Date: 06 Oct 1995

Jacob Nielsen writes: I don't know of any TeX macros that handles prettyprinting by it self,

You could take a look at the paper "Pascal prettyprinting: an example of `preprocessing within TeX'" by Jean-Luc Doumont, published in TUGboat Vol 15 No 3, Sept 1994 (1994 Annal Meeting Proceedings) pages 302--308. From the abstract: "Prettyprinting a piece of Pascal code with TeX is often done via an external preprocessor. Actually, the job can be done entirely in TeX; this paper introduces PPP, a Pascal pretty-printer environment that allows you to typeset Pascal code by simply typing \Pascal <Pascal code> \endPascal. The same approach ... can be applied easily and successfully in numerous other situations."

This seems to be pretty much what the question was about, except that it is for Pascal rather than C++ or FORTRAN. So there is still some work to do here...

BTW, I think that the TeX teletype is quite nice in any case :-)

To use `teletype' in combination with TeX is quite an anachronism. Wherever TeX uses `tt' (never `tty'!), like in \tt or the font name cmtt10 it stands for `typewriter type', a family of fonts designed to imitate the text produced by a classical typewriter (monospace, uniform stroke width, rounded stroke endings). The teletype is a particular brand of hardcopy terminals (does anybody still know what that is?) that was probably already obsolete in all but the oldest implementations of UNIX, which has nevertheless kept the term alive by using the indication `tty' for arbitrary terminal ports. I hope this may help to reduce the widespread confusion between these terms.

Publishing forums

From: **Kiyoshi Akima**
Date: 13 Oct 1995

Is there a journal or other forums that publish literate programs on a regular basis? Not necessarily articles and papers on the subject, but actual programs? I'm looking to read some programs, besides the examples that come with the various systems. Of course, I know that somebody named Knuth published a few of his literate programs, but I'm looking for more. :-)

From: **Kevin Purcell**
Date: 13 Oct 1995

The book "Literate Programming" collected a few programs (published elsewhere together). The CACM did it for several years until realising that literate programming seemed to be turning into tools wars (noweb seems to be reversing the trend). Papers pop-up in various different places (mostly educational or reports of work done), e.g. IEEE Software and Software: Practice and Experience. As far as I know there is no "Journal of Literate Programming". If you find any more information post it to the literate programing list/newsgroup.

From: **Fabricio Chalub Barbosa do Rosario**
Date: 16 Oct 1995

Kevin Purcell writes: The CACM did it for several years until realising that literate programming seemed to be turning into tools wars (noweb seems to be reversing the trend).

I'm sorry... I didn't get the point, what do you mean by tools wars ?

From: **Kevin Purcell**
Date: 16 Oct 1995

Fabricio Chalub Barbosa do Rosario writes: I'm sorry... I didn't get the point, what do you mean by tools wars ?

Chris Van Wyk (the moderator of the literate programing colum in the CACM) pointed out in one of his columns that very few people had done any literate programming with a tool that they had not built themselves and that the field seemed to have deteriorated into the proliferation of literate programming tools.

When writing a source editor that checked pre- and post-conditions, Norman Ramsey noted that: 1. A lot of the features in WEB were not used. 2. Prettyprinting the source to match the developers desires was difficult. 3. The TeX source was not very human readable. He wrote this up in Software: Practice and Experience.

I beleive this (and the feeling that people should be easily able to experiment with literate programming tools) lead him to create noweb with the feature set that it has and the implmentation that it has (built as a pipeline with replaceable parts) so that users can experiment with it. Apologies for the lack of references here, but I was in the UW library reading these papers over the weekend. I casn fish out the references if anyone is interested.

From: **Felix Gaertner**
Date: 16 Oct 1995

Kiyoshi Akima writes: Is there a journal or other forums that publish literate programs on a regular basis? Not necessarily articles and papers on the subject, but actual programs? I'm looking to read some programs, besides the examples that come with the various systems.

Unfortunately I know of no such publishing forums for literate programs. But I am convinced that such a forum would be a very good thing. Of course there are a lot of literate programs around that you can print out and read. But after printing out 100 pages you often notice that the program is not worth reading at all. A publishing forum for literate programs could establish a base set of literate programs that are really worth reading (i.e. they are interesting, instructive and entertaining).

I would like to hear others' opinions on what programs they consider worth reading. Of course, you would put TeX and Metafont first on the list. Then there are a couple of instructive examples published in the CACM column `programming pearls' in the mid 80s (I don't have the exact references handy at the moment). What else?

From: **Matthias Neeracher**
Date: 17 Oct 1995

Felix Gaertner writes: Unfortunately I know of no such publishing forums for literate programs. But I am convinced that such a forum would be a very good thing. Of course there are a lot of literate programs around that you can print out and read. But after printing out 100 pages you often notice that the program is not worth reading at all.

I'm currently trying an experiment. I'm trying to present a "Work in Progress". GUSI is a BSD style socket library for the Apple Macintosh, written in C++, that I'm currently rewriting and reengineering in noweb. I will try to upload new versions of my work every few days or weeks, so readers of the page can see the progress of the work.

Unfortunately, I'm not exactly the world's greatest literate programmer, but I hope that the code at least shows off noweb's HTML capabilities and illustrates the style of C++ web design that I've drifted to. Due to the online nature of the work, you can also browse it without having to print anything out.

Publishing programs on WWW is of course not the sort of quality building resource that a peer reviewed journal could be, but people could start building "Meta-pages" with their links to and comments on other people's literate programs. Maybe we should even build in a facility for section-by-section annotation?

CWEB & C++ trouble with `const'

From: **Jan Dvorak**
Date: 05 Nov 1995

I'm using CWEB for C++ programming. It works generally fine; there is, however, one annoying little detail about the way CWEAVE and TeX format the `const' keyword. There is a slight difference over plain C: in C++ a `const' can stand also after a function declaration, as in

```
class Location {@/
  @<internals of |class Location|@>@;
public:@/
   double GetX() const;
   double GetY() const;
};
```

A step-like output will be produced from this source:

```
   double GetX()
        const;
        double GetY()
             const; };
```

which is rather disturbing. It looks to me that `const' is handled as a type specifier, pretty much as `int' or `double', and CWEAVE then expects arguments in the old K&R style. I tried to use @s to change the formatting category of const, but there seems to be no good one. Anybody had got a similar experience? Could I change this behavior without much surgery in CWEAVE or cwebmac.sty?

From: **Jan Dvorak**
Date: 06 Nov 1995

Thanks to Andreas Scherer for his reply: It seems to me that you're stuck with an old version of CWEB. I copied these few lines to a `test.w' file and padded it with the obvious elements to make it run through CWEAVE, and alas, the result is a faithful image of the input. The programs installed on my machine are CWEAVE 3.4d and TeX 3.14159.

Well, I checked my version of CWEAVE, it says 3.4. I got it from the CTAN some two months ago, so I don't suppose this would be much of a problem. However, I don't know how to see the patchlevel (you write 3.4d), the installation directories have been purged. But read on, there comes a real life example. Admittedly, the above example was not thought through too well --- it worked for me correctly as well. I'm sorry for that. Now, there is a real example where I do get the weird formatting:

```
% -- CWEB file begins here

\def\twodim/{2-\kern-0.1ex D}

@* Matrices in \twodim/.
@s Matrix2D class
@c
@<General matrices in \twodim/@>@;

@ @<General matrices in \twodim/@>=
class Matrix2D
{
public:@/
```

```
  Matrix2D(double a11, double a12, double a21, double a22);
  Matrix2D(const Vec2D& row1, const Vec2D& row2);
  Matrix2D(const SymMatrix2D& smat);

  Matrix2D  operator +  (const Matrix2D& a) const;
  Matrix2D& operator += (const Matrix2D& a);
  Matrix2D  operator -  () const;
  Matrix2D  operator -  (const Matrix2D& a) const;
  Matrix2D& operator -= (const Matrix2D& a);
  Matrix2D  operator *  (const double r) const;
  Matrix2D& operator *= (double r);
  Matrix2D  operator *  (const Matrix2D& a) const;  // Matrix product
  Matrix2D  operator /  (double r) const;
  Matrix2D& operator /= (double r);

  double trace() const;
  double inner_product(const Matrix2D& a) const;

public:@/
  double A[4];@/
};
@<Inlines for general matrices in \twodim/@>@;

@ @<Inlines for general matrices in \twodim/@>=
Matrix2D::Matrix2D(double a11, double a12, double a21, double a22)
{@+ A[0] = a11, A[1] = a12, A[2] = a21, A[3] = a22; @+}@#

Matrix2D::Matrix2D(const Vec2D& row1, const Vec2D& row2)
{@+ A[0] = row1.X, A[1] = row1.Y, A[2] = row2.X, A[3] = row2.Y; @+}@#

Matrix2D::Matrix2D(const SymMatrix2D& smat)
{@+ A[0] = smat.A[0], A[1] = A[2] = smat.A[1], A[2] = smat.A[3]; @+}@#

Matrix2D  Matrix2D::operator + (const Matrix2D& a) const@/
{@+ return Matrix2D(A[0]+a.A[0],A[1]+a.A[1],A[2]+a.A[2],A[3]+a.A[3]); @+}@/

Matrix2D& Matrix2D::operator += (const Matrix2D& a)@/
{@+ A[0]+=a.A[0],A[1]+=a.A[1],A[2]+=a.A[2],A[3]+=a.A[3]; @+}@#

Matrix2D  Matrix2D::operator - () const@/
{@+ return Matrix2D(-A[0],-A[1],-A[2],-A[3]); @+}@#

Matrix2D  Matrix2D::operator - (const Matrix2D& a) const@/
{@+ return Matrix2D(A[0]-a.A[0],A[1]-a.A[1],A[2]-a.A[2],A[3]-a.A[3]); @+}@#

Matrix2D& Matrix2D::operator -= (const Matrix2D& a)@/
{@+ A[0]-=a.A[0],A[1]-=a.A[1],A[2]-=a.A[2],A[3]-=a.A[3]; @+}@#

Matrix2D  Matrix2D::operator * (const double r) const@/
{@+ return Matrix2D(r*A[0],r*A[1],r*A[2],r*A[3]); @+}@#

Matrix2D& Matrix2D::operator *= (double r)@/
{@+ A[0]*=r,A[1]*=r,A[2]*=r,A[3]*=r; @+}@#

Matrix2D Matrix2D::operator * (const Matrix2D& a) const@/
{   return Matrix2D(A[0]*a.A[0]+A[1]*a.A[2],@|
                    A[0]*a.A[1]+A[1]*a.A[3],@|
                    A[2]*a.A[0]+A[3]*a.A[2],@|
                    A[2]*a.A[1]+A[3]*a.A[3]);
}@#

Matrix2D Matrix2D::operator /  (double r) const@/
{@+ return Matrix2D(A[0]/r,A[1]/r,A[2]/r,A[3]/r); @+}@#

Matrix2D& Matrix2D::operator /= (double r)@/
{@+ A[0]/=r,A[1]/=r,A[2]/=r,A[3]/=r; @+}@#

double Matrix2D::trace() const@/
```

```
{@+ return A[0]+A[3]; @+}@#

double Matrix2D::inner_product(const Matrix2D& a) const@/
{@+ return A[0]*a.A[0] + A[1]*a.A[1] + A[2]*a.A[2] + A[3]*a.A[3]; @+}@#

@ Index.

% -- CWEB file ends here
```

You tried? My output is generally good in the class declaration section, there is only one mistake: the two lines above the // Matrix product comment come out concatenated. The real troubles start in the @<Inlines ...@> section. It may be caused by a combination of the `operator' keyword with the `const'. The result is that the body of the first operator function

```
   Matrix2D  Matrix2D::operator + (const Matrix2D& a) const@/
   {@+ return Matrix2D(A[0]+a.A[0],A[1]+a.A[1],A[2]+a.A[2],A[3]+a.A[3]); @+}@#
```

gets indented (and it would do the same without those @+ by the braces). And then, the following function starts at the indented horizontal position but, since this one has no const at the end, does not add to the indentation. Then there comes the unary minus operator the again has a const at the end: and it indents its body as before. And so on ...

Yes, the last two methods, trace() and inner_product(...), do not contribute to the odd behavior. So the outcome is that

```
   CWEAVE and TeX format the `operator ...() const' construction incorrectly.
```

By the way, without those @/ at the ends of the member function header lines, one ends up with the opening brace at the end of the header line. And without those @# at the ends of the member function bodies, one would get a goulasch (= a perfect disorder). Of course, I could write an @<Inlines ...@> section for each function, and I'll have to do it unless another way out appears. But that's not what I would be too happy with -- I will probably never want to add any comments to member functions of a class that is this simple, so I'd be just wasting paper. Thanks for any upcoming comments...

From: **Marc van Leeuwen**
Date: 08 Nov 1995

Jan Dvorak writes: Now, there is a real example where I do get the weird formatting:

```
% -- CWEB file begins here

\def\twodim/{2-\kern-0.1ex D}

@* Matrices in \twodim/.
@s Matrix2D class
@c
@<General matrices in \twodim/@>@;

@ @<General matrices in \twodim/@>=
class Matrix2D
{
public:@/
  Matrix2D(double a11, double a12, double a21, double a22);
  Matrix2D(const Vec2D& row1, const Vec2D& row2);
  Matrix2D(const SymMatrix2D& smat);

  Matrix2D  operator +  (const Matrix2D& a) const;
  Matrix2D& operator += (const Matrix2D& a);
  Matrix2D  operator -  () const;
  Matrix2D  operator -  (const Matrix2D& a) const;
  Matrix2D& operator -= (const Matrix2D& a);
  Matrix2D  operator *  (const double r) const;
  Matrix2D& operator *= (double r);
  Matrix2D  operator *  (const Matrix2D& a) const;  // Matrix product
  Matrix2D  operator /  (double r) const;
  Matrix2D& operator /= (double r);

  double trace() const;
  double inner_product(const Matrix2D& a) const;

public:@/
  double A[4];@/
```

```
};
@<Inlines for general matrices in \twodim/@>@;

[ another CWEB section omitted here ]
```

You tried? My output is generally good in the class declaration section, there is only one mistake: the two lines above the // Matrix product comment come out concatenated.

A first observation is that `@s Matrix2D class' is a dangerous thing to say, since it is a lie (and you know what will happen if you lie to a compiler): the type name `Matrix2D' should not behave like `class' (or `struct'), but rather like `int'. In this example you get away with this since the grammar happens to reduce all occurrences of `Matrix2D' in a reasonable way, but always for the wrong reasons, and a cast like |(Matrix2D*)| would cause real trouble.

That being said I'll tackle your example now (I will see if I get to understand your even more problematic other section, which was omitted above, some other time). Although you call the output ``generally good'', close inspection shows that some strange things are going on. The indentation of the lines with `class' and with the `};' is more than for `public', which is an indication that the outer structure did not get recognised correctly due to a parsing error; this can be made manifest by adding `@1' in the code, which will print one irreducible scrap sequence that includes the `class', its braces, and some unparsable scraps inside. This is a direct consequence of the problem leading to the concatenation of lines that you mentioned (in fact the syntax rule that should have forced a line break failed to come into action on that place), so in itself this is not too surprising. However it is strange that the spacing after `operator' is different for the three cases where `*' follows `operator' than for all other cases. The second one with `*=' looks particularly bad, with a space between the two tokens. To understand this, calls for some heavy debugging (level @2); I did this and will try to explain my findings. What follows is very technical, and incomprehensible unless you have the table of CWEB grammar rules at hand (and in fact the implementation of those rules further on in CWEAVE, since this is not always reflected accurately in the table); I provide it mainly to demonstrate just what an awful mess the Levy/Knuth grammar is.

The general plan with an |operator_like| keyword is that it combines with the following operator token to form an expression, which is stated in rule 97 (actually, the operator token has one of the categories |binop|, |unop|, or |unorbinop|). This seems simple enough, but there is a complication: in cases like `operator +=', we do not want to combine `operator' with `+', but first `+' with `=', by rule 19. I can only guess why this matter was not dealt with at the lexical scanning level; in bad old K&R C such compound operators used to be two separate tokens, so that you could put white space or even whole comments in between (but if you try to put a comment there in CWEB, or even *after* the `=', you will run into another kind of trouble, caused by a bug that I don't wish to discuss here). In any case we have a minor problem with `operator +=', since the reduction engine of CWEAVE gives precedence to the leftmost matching rule, so it would combine `operator+' (97) before it can combine `+=' (19). To repair this, there is a footnote to rule 97 that says it is not applied if the next token is |binop| (the category of `='). However, there is another danger, namely rule 99, which makes |operator_like| decay to |new_like| if it is not matched by any other rule. Even if rule 97 is inhibited by its footnote, rule 99 matches more leftwards than rule 19 for `+=', and I was truly puzzled why the debugging output showed that rule 99 is not applied in this case. The truth of the matter cannot be deduced from the table, but only from its implementation: in reality if rule 97 appears to match but is then blocked by its footnote, no other rules for |operator_like| are tried (a |break| is executed which jumps right out of this case of the surrounding switch). To describe this properly, rule 99 should have a footnote explaining that if 97 does not match, but only because of its footnote, then rule 99 does not match either. In any case CWEAVE manages to reduce `operator +=' properly to an expression.

What about `operator *'? This seems simpler, but there is a catch: the category of `*' is not |unorbinop| initially, but rather |raw_unorbin|; this has to do with other quirks like `*const' (rule 105). Now in our example there is no `const' after the `*', and rule 106 tells |raw_unorbin| to decay to |unorbinop|, so is there a problem? Yes there is, because |operator_like| is also eager to decay (to |new_like|) and being to the left, it has precedence. So what happens is this: rule 97 does not apply, but not because of its footnote but rather because |raw_unorbin| is not one of |binop|, |unop|, or |unorbinop|. Therefore the undocumented exception to rule 99 does not apply and |operator_like| decays to |new_like|, making it impossible for rule 97 to match any more. Even then |raw_unorbin| is not allowed to decay to |unorbinop|, because rule 94 catches it early, and combines it with the preceding |new_like| into another |new_like| (rule 94 was clearly intended for some other purpose, although I cannot figure out exactly which, but it is applied nonetheless). By now the parser is really screwed up, and it is a bit of a surprise that it manages to recover, with the visible damage limited to some spurious space around the `*'. The explanation is that `(const double r) const' becomes a |cast| (as it should) after which rule 95 combines it with the preceding |new_like| to an |exp| after which the air is cleared; again rule 95 was intended for rather different situations.

So finally we come to the declaration of `operator *=', which does cause to parser to jam. Why? Well, like for `operator *' (but contrary to `operator +=', rule 99 is not shielded by rule 97, so `operator' becomes |new_like|. Then rule 94 again combines this with the `*'; unlike rule 97 no exception is made if an `=' follows, like it does here. This leaves the `=' separated from its `*', and in fact it fails to combine with anything else in the remainder of the parse, which also prevents the current declaration to be recognised. This explains why CWEAVE outputs the `=' surrounded by spaces, and why it fails to insert a line break (usually produced by rule 40) in front of the declaration containing it.

As you can see, analysing a single innocent looking failure open up a cesspool of ugly details of the CWEB grammar (which, I should like to add, are partly due to the fact the the official C++ syntax is a complete mess, if it is properly defined at all). I can imagine that such experiences would frustrate some people so much that they get convinced that prettyprinting itself is a bad idea; however I do not think this conclusion is justified by the fact that CWEB comes with a lousy prettyprinting grammar, just like it is not justified to dismiss programming in general just because some extremely bad programs have been written, including some that are very widely used.

From: **Marc van Leeuwen**
Date: 09 Nov 1995

Since I had debugged with the `@s Matrix2D class' line still in there, I did not realise that your your problems nearly disappear if you change it to `@s Matrix2D int'. To be precise, you will have no irreducible scrap sequence any more in section 2 (which I discussed at length in the previous message), and only one irreducible scrap sequence in section 3 (which I did not discuss, but where you had a major problem); moreover the output looks superficially correct in both cases, and you would probably not have complained if you had got this output. Even better, none of the `@#' codes you placed in section 3 are necessary with this change, except the one before the inline definition of `operator *=' (see below). So one might say that your problems were really CWEAVE's revenge for your `@s Matrix2D class'.

Despite this rehabilitation of CWEAVE, most of the analysis of my previous message still stands, and if you look closely, you can see that `operator *' and `operator *=' are formatted in a different manner than `operator +' and `operator +='. In particular in each of the instances of `operator *=' (in sections 2 and 3) the `=' gets separated from the `*' it belongs to. In the first instance however, the follwing |cast| `(double r)' first reduces to |exp| by rule 22, and then rule 36 incorporates the `=' together with the |decl_head| before it and the |exp| after it, so that parsing may continue; note that the construction is treated as if it were a declaration of `operator *', initalised by the `(double r)', which is of course nonsense. In the second instance on the other hand, there is no `;' following the |cast| `(double r)', so rule 22 does not apply, and the isolated `=' remains unmatched until the end, where it ends up in an irreducible scrap sequence. It is a bit of luck that the parts before and after the `=' reduce reasonably well on their own, so that apart from the spaces around `=' the only visible defect is the lack of vertical white space before the declaration, assuming that all explicit codes `@#' are removed.

I did not study why section 3 had increasing and unmatched indentation levels with `@s Matrix2D class' in place, since it is not really surprising that unsyntactical input produces strange output. One thing is clear from the grammar however, namely that indents are usually not produced by the same rules as the matching outdent; therefore unmatched indentation levels are not really surprising, especially in the presence of irreducible scrap sequences.

From: **Jan Dvorak**
Date: 09 Nov 1995

Marc van Leeuwen writes: A first observation is that `@s Matrix2D class' is a dangerous thing to say, since it is a lie (and you know what will happen if you lie to a compiler): the type name `Matrix2D' should not behave like `class' (or `struct'), but rather like `int'. In this example you get away with this since the grammar happens to reduce all occurrences of `Matrix2D' in a reasonable way, but always for the wrong reasons, and a cast like |(Matrix2D)| would cause real trouble.*

I see. Yes, this may be the cause of the whole trouble.

That being said I'll tackle your example now (I will see if I get to understand your even more problematic other section, which was omitted above, some other time). Although you call the output ``generally good'', close inspection shows that some strange things are going on. The indentation of the lines with `class' and with the `};' is more than for `public', which is an indication that the outer structure did not get recognised correctly due to a parsing error; this can be made manifest by adding `@1' in the code, which will print one irreducible scrap sequence that includes the `class', its braces, and some unparsable scraps inside. This is a direct consequence of the problem leading to the concatenation of lines that you mentioned (in fact the syntax rule that should have forced a line break failed to come into action on that place), so in itself this is not too surprising. However it is strange that the spacing after `operator' is different for the three cases where `' follows `operator' than for all other cases. The second one with `*=' looks particularly bad, with a space between the two tokens. To understand this, calls for some heavy debugging (level @2); I did this and will try to explain my findings. What follows is very technical, and incomprehensible unless you have the table of CWEB grammar rules at hand (and in fact the implementation of those rules further on in CWEAVE, since this is not always reflected accurately in the table); I provide it mainly to demonstrate just what an awful mess the Levy/Knuth grammar is. [the VERY technical discussion is snipped out]*

Ouuff! Yes, it is a terrible mess.

As you can see, analysing a single innocent looking failure open up a cesspool of ugly details of the CWEB grammar (which, I should like to add, are partly due to the fact the the official C++ syntax is a complete mess, if it is properly defined at all). I can imagine that such experiences would frustrate some people so much that they get convinced that prettyprinting itself is a bad idea; however I do not think this conclusion is justified by the fact that CWEB comes with a lousy prettyprinting grammar, just like it is not justified to dismiss programming in general just because some extremely bad programs have been written, including some that are very widely used.

Well, I must confess, I was very close to throwing the whole CWEB away and switching to something without any prettyprinting, like nuweb or noweb. I installed nuweb, rewrote the Matrix2D example into it, ran it through and -- returned back to CWEB. I probably got spoiled by the nice-looking output it produces -- almost always. Many thanks for the work you put in this small-looking affair.

From: **Jan Dvorak**
Date: 09 Nov 1995

*Thanks to Marc van Leeuwen for his extensive replies. Since I had debugged with the `@s Matrix2D class' line still in there, I did not realise that your your problems nearly disappear if you change it to `@s Matrix2D int'. To be precise, you will have no irreducible scrap sequence any more in section 2 (which I discussed at length in the previous message), and only one irreducible scrap sequence in section 3 (which I did not discuss, but where you had a major problem); moreover the output looks superficially correct in both cases, and you would probably not have complained if you had got this output. Even better, none of the `@#' codes you placed in section 3 are necessary with this change, except the one before the inline definition of `operator *=' (see below). So one might say that your problems were really CWEAVE's revenge for your `@s Matrix2D class'.*

Yes, revenge is the right word here. The trouble is, there are not that many C++ examples with const operator overloading and const in CWEB to study. (And I did go through your necklace, String2 and IntArray examples).

*Despite this rehabilitation of CWEAVE, most of the analysis of my previous message still stands, and if you look closely, you can see that `operator *' and `operator *=' are formatted in a different manner than `operator +' and `operator +='. In particular in each of the instances of `operator *=' (in sections 2 and 3) the `=' gets separated from the `*' it belongs to. In the first instance however, the follwing |cast| `(double r)' first reduces to |exp| by rule 22, and then rule 36 incorporates the `=' together with the |decl_head| before it and the |exp| after it, so that parsing may continue; note that the construction is treated as if it were a declaration of `operator *', initalised by the `(double r)', which is of course nonsense. In the second instance on the other hand, there is no `;' following the |cast| `(double r)', so rule 22 does not apply, and the isolated `=' remains unmatched until the end, where it ends up in an irreducible scrap sequence. It is a bit of luck that the parts before and after the `=' reduce reasonably well on their own, so that apart from the spaces around `=' the only visible defect is the lack of vertical white space before the declaration, assuming that all explicit codes `@#' are removed.*

I could fix the appearance of the *= operator by surrounding it with @[@]. As a matter of fact, I did this to all my operators, to be consequent. Thanks to Andreas Scherer and Lee Wittenberg for their suggestions.

I did not study why section 3 had increasing and unmatched indentation levels with `@s Matrix2D class' in place, since it is not really surprising that unsyntactical input produces strange output. One thing is clear from the grammar however, namely that indents are usually not produced by the same rules as the matching outdent; therefore unmatched indentation levels are not really surprising, especially in the presence of irreducible scrap sequences.

My experience is that the `butterfly effects' are very strong here. You forget a semicolon and then get a mess three pages later. And I often do forget about these small things when I'm prototyping a program. All works well now. Regarding those examples that are publicly available: Do you think it is easier to do C++ literate programming using FWEB?

From: **Lee Wittenberg**
Date: 09 Nov 1995

*Jan Dvorak writes: I could fix the appearance of the *= operator by surrounding it with @[@].*

Personally, I prefer the "alternate" op= macros (that use \KK, I think) in cwebmac.tex to the "standard" ones that seem to produce problems in your output.

From: **Roland Kaufmann**
Date: 19 Nov 1995

Without wanting to offend anyone, but the detailed discussion of the shortcomings of CWEB's prettyprinting grammar makes me wonder whether it is worth doing (prettyprinting, of course). I personally try to keep my source (i.e. web) indented, so noweb should do a reasonable job on it, maybe one could use different fonts for keywords (I gather that there are filters for noweb to do that), isn't that enough? Another interesting observation is that the CWEB parser is hand-made, (as are SpiderWeb's according to my [limited] knowledge, don't know about FWEB) whereas most compiler use automatically generated parsers (for good reasons, I suppose) are there any technical (as opposed to historical, personal habits, etc.) reasons why the prettyprinting grammars are made (or woven?) by hand?

Announcement of MetaPost

From: **John Hobby**
Date: 01 Apr 1992

I am the author of MetaPost. In response to your query, I am enclosing a copy of an anouncement that should be coming out soon in the TeXhax mailing list. I will post a similar item to comp.text.tex. Feel free to contact me if you would like more information.

At the TeX User's Group Meeting in August 1989, I first announced my plans to create a picture-drawing language based on METAFONT but with PostScript output and features for dealing with text as well as graphics. By the summer of 1990, I had a preliminary version installed on a few machines here at Bell Labs, but I was not able to make it available externally. Since then, I have perfected the language implementation and the support software and I have written a User's Manual, but I had trouble getting permission to make the software available outside of AT&T.

Now I can finally announce that the MetaPost language implementation is available free to any academic institution. It is still necessary to sign a non-disclosure agreement. Interested parties should contact me at the address below and give an address where I can have the necessary paperwork sent.

Here are the main features of MetaPost:

1. The language is based on METAFONT and has almost the same syntax and semantics. This includes data types for curves, pictures, and linear transformations, as well as powerful macros and the ability to solve linear equations.

2. Pictures can contain typeset text using any font for which you have a .tfm file. MetaPost input can also contain text to be typeset with TeX (or troff). There are predefined macros for things like positioning labels and finding bounding boxes.

3. There are primitives and predefined macros for accessing PostScript features such as colors and shades of gray, dashed lines, and clipping, and for controlling the appearance of corners and ends of lines.

While it is possible to use MetaPost to generate Type 3 PostScript fonts, the intended application is generating figures for technical documents. For this reason, MetaPost output consists of PostScript files that can be merged with TeX output using graphics inclusion features in dvi-to-PostScript translators.

Since many dvi-to-PostScript translators download TeX fonts by including only those characters that are actually used by the .dvi file, extra provisions are necessary to allow such fonts in PostScript files generated via MetaPost. Version 5.47 of Tom Rokicki's dvips contains undocumented features for doing this, but a few bug-fixes are necessary to make them work properly; versions 5.481 and higher should have the feature documented and working properly. You can also use any dvi-to-PostScript program that understands encapsulated PostScript with standard comments like "%%DocumentFonts: cmr10".

The actual MetaPost distribution contains web source for the translator and auxiliary programs, change files for use with the standard UNIX (r) TeX distribution (for use with web2c), auxiliary routines written in C, and a shell script that controls the preprocessing of TeX commands found in MetaPost input files. I may also include the bug fixes for dvips 5.47.

Announcement of CLiP

From: **Eric van Ammers**
Date: 24 Feb 1993

CLiP, A Universal Literate Programming Tool

Eric. van Ammers

Abstract

CLiP (Code from Literate Program) is a tool which allows writing literate programs in virtually any programming language (Pascal, Fortran, C, C++, Assembler languages, etc) and in combination with almost any formatter (Runoff, Troff, TeX, LaTeX, etc.) or word-processor (Winword, Lotus Manuscript, Ami Pro, Word Perfect, Wordstar, etc). This posting explains the CLiP philosophy and compares it to the WEB approach. CLiP turns out much more general. In spite of this generality its disadvantages as compared to the WEB-family are few indeed. Currently we have two versions of CLiP, for VAX/VMS and for MS-DOS platform. Although only limited user documentation is available, this is not felt as a big problem since CLiP works fairly intuitively.

1 Introduction It is clearly impossible to realise literate programming without a supporting tool. Historically Knuth was the first to report the very idea of literate programming using his WEB-system and most literate programming practitioners today employ WEB or one of its derivatives. The original WEB philosophy was to provide a literate programming tool for a particular programming language (Pascal) in a particular formatting environment (TeX). Consequently a whole family of WEB's have emerged to satisfy the needs of individuals who wanted to program literately on different platforms. However, the idea to extract compilable modules from documented

refinement steps (rather then to create modules and documentation separately) has independently originated elsewhere too. Not surprisingly, the corresponding literate programming tools have been based on different principles and show rather distinct characteristics. At the Wageningen Agricultural University e.g. we have developed VAMP (1984) and later CLiP (1992) and these tools show interesting differences as compared to the WEB-family [1,2,4,5]. Because the introduction of CLiP made VAMP obsolete, we will focus our attention to the CLiP-system. First we explain the design philosophy of CLiP and we briefly sketch the way CLiP works. Next we describe the major differences between the CLiP-system and the WEB-family. These differences are mostly due to the difference in philosophy of both systems. Then we report on the status of the CLiP project and we conclude by a sketch of our activities in the near future. A more extensive description of CLiP is in [5].

2 Design philosophy CLiP (and formerly VAMP) was designed from the idea that "good" programming has little or nothing to do with programming languages. We consider stepwise refinement a "good" programming technique. So when we decided to build a tool that would allow the extraction of modules from documented refinement steps, it was evident this tool would have to operate independent of the programming language involved. In addition this approach would be beneficial from the point of view of maintenance. The latter perspective made us decide to design the tool also as much as possible independent of formatter or word-processor. According to the CLiP approach the extracted modules definitely have a function and should not be "deliberately unreadable" as Knuth proposes [2]. Since compiler and debugger messages relate to derived modules rather than to the documentation proper, it should be easy to relate the code lines of the generated modules to the corresponding documentation lines. For this reason we want to copy the code lines of the documentation unchanged into the modules.

3 How CLiP works A literate programming tool has to extract modules from input files (called "sourcefiles") that serve as (input for) documentation at the same time. The first problem for an literate programming tool to solve is to separate text segments (that are meant as informal explanation) from code segments (that contain the actual code to be extracted). The second problem is to merge the code segments into output files (called "modules"). For this purpose we have in a conventional literate programming environment (like WEB or VAMP) special command lines that are added to the sourcefiles. The command lines control the extraction of the modules by the module generator but are ignored by the formatter. This technique does not work if the documentation is processed by a word-processor, since the command lines would invariably show up as lines in the documentation, which is highly undesirable. CLiP solves the problem by prescribing a special programming style. Its input files are either obtained directly by an editor or indirectly by an ASCII-export from a word-processor. CLiP recognizes special comment lines as indicators to guide the module extraction process. These comments look 'natural' in the context of the code. The syntax CLiP recognizes is parameterized and can be adjusted to virtually any programming language.

4 CLiP compared to the WEB-family The differences between the CLiP-system and the WEB-family are partly due to general design decisions and partly to the difference in philosophy.

4.1 General design differences

- CLiP works not "monolithic" (like WEB does). It processes up to 64 (the number can be adjusted) input files in one run. From this input it produces as many modules as are specified by the user. Thus it is possible to generate a complete software system in one single CLiP-run.
- CLiP composes modules from stubs which may be scattered over multiple source files.
- CLiP allows a global redefinition of stubs. In this way one can temporarily put a stub in a given slot and replace it later on. This feature makes it possible to define abstraction levels in the description of a system. For instance one can introduce a particular record structure at a higher level as a simple name with only its most important fields and defer the definition of the other fields to a suitable lower level. Such a form of data abstraction is known as "partially specified data structures".
- Unlike WEB, CLiP has no macro facilities.

4.2 Programming language independence CLiP will work seamless with any programming language that allows comment lines between the tokens of the language. Otherwise its applicability can in principle be restricted, but we know of no language where this would be a problem in practice. Since CLiP is completely programming language independent, it has no knowledge of the programming language it is processing. Thus it will not recognize keywords, identifiers or other tokens.

- CLiP cannot automatically produce a X-reference list of identifiers the way WEB does (in this respect CLiP is definitely less powerful than WEB). With CLiP a X-reference list must be produced the same way as an index of an ordinary document. This feature is therefore highly dependent of the particular formatter or word-processor one applies. But with a modern word-processor like WinWord or Ami Pro, powerful tools exist to support the construction of an index.
- CLiP can extract any sort of file from the documentation. So all kind of additional files can be documented also, rather than the pure code only. One can think of files containing the error message templates of a system, batch files, internal tables that are present as a file, etc.
- The special lines that CLiP recognizes can be adjusted to suit virtually any programming language. However, the system operates strictly on a line basis.
- CLiP allows a fine-tuning of the module generation process by means of "options". But these are cosmetical and will not be discussed here.
- Unlike WEB, CLiP has no compiler like knowledge and it does not extend the programming language one uses in any way. Nor does it compensate any nasty features.

4.3 Formatter and word-processor independence CLiP simply processes all the lines that are enclosed between a special type of comment lines it recognizes. Such segment should contain only code. CLiP copies the lines from the sources into the modules without

any formatting (i.e. "verbatim" or "literal"). This means that CLiP will cooperate with any formatter that has a command like "verbatim" or "literal" (all formatters that we know off do have such a command). CLiP will process the same files that otherwise would be formatted. In a word-processor environment it is required that the word-processor has an adequate ASCII-export, which eliminates formatting information. CLiP will analyse the ASCII-files rather than the original word-processor files and generate the modules from there. Again we do not know of any word-processor where there could be a problem in this respect.

- The documentation of refinement steps using CLiP is entirely free and only limited by the text processing system one is using. No order is imposed for the refinements nor any hierarchy in terms of sections and subsections (WEB is fairly restricted here).
- No restrictions exist, other than the limitation of the particular word-processing system one uses, to explain the program that is documented. Illustration by means of tables, diagrams, figures or pictures are no problem.
- CLiP generates modules that strongly resemble the code one finds in the documentation. This is convenient for the programmer who wants to use them for debugging purposes and the like. Although this does not really solve the so called "preprocessor problem", it makes it a lot easier to live with than in a WEB-environment [3].

5 Project status Currently CLiP experiences its second version which exists for VAX/VMS (written in VAX-Pascal) and for MS-DOS (written in Turbo Pascal Vision). Both systems are of course documented as CLiP literate programs themselves. The user documentation of CLiP currently consists of a short description of how CLiP works and should be used. Although it definitely does not have the status of a manual, it should allow programmers to get along with the CLiP-system. Currently we only have a limited number of examples and demos which moreover are fairly trivial. Better ones are on the priority list. Of course we have real systems build with CLiP (e.g. CLiP itself is a literate program in CLiP) but they are too complex to qualify as useful examples.

6 Future activities From the discussion inside the literate programming group we infer that CLiP, due to its original design philosophy, may be a valuable addition to the set of literate programming tools. Our goal is to make CLiP as quickly as possible available to the literate programming audience by means of anonymous FTP. We aim at the following time schedule:

1. Executable versions of CLiP for VAX/VMS and MS-DOS + provisional operating manual + trivial example program will be FTP-able by March 15, 1993.

2. More extensive example programs in different programming languages will follow incrementally in the successive months.

3. We are looking for an opportunity to have CLiP ported to Unix. Resources for a job like this are currently extremely scarce at our university, and unfortunately we are unable to set a date yet. External help would be very welcome.

7 References

1. Ammers E.W. van et.al. 1984. "VAMP: A Tool for Programming by Stepwise Refinement". Internal report. Department of Computer Science, Wageningen Agricultural University.

2. Knuth D.E., 1984. "Literate Programming". The Computer Journal 27, 2, pg. 97-111.

3. Ramsey N., Marceau C. 1991. "Literate Programming on a Team Project". Software Practice and Experience 21, 7, pg 677-683.

4. Ammers E.W. van, Kramer M.R. 1992. "VAMP: A Tool for Literate Programming Independent of Programming Language and Formatter". CompEuro '92 Proceedings, May 4-8 1992, the Hague, pg. 371-376.

Eric W. van Ammers

Wageningen Agricultural University

Department of Computer Science

Dreijenplein 2

6703 HB Wageningen

The Netherlands

Voice: +31 (0)8370 83356/84154

Fax: +31 (0)8370 84731

E-mail: ammers@rcl.wau.nl

From: Joachim Schrod
Date: 25 Feb 1993

CLiP works not "monolithic" (like WEB does). It processes up to 64 (the number can be adjusted) input files in one run. From this input it produces as many modules as are specified by the user. Thus it is possible to generate a complete software system in one single CLiP-run.

Two questions: (1) Where is WEB "monolithic"? (Please don't answer me that Pascal is monolithic. I ask where WEB itself is monolithic.) (2) I get the impression that in the text above the ability to process only one file at time is synonym to "monolitic". Is this correct? If yes, this would mean that a C++ compiler who can process only one C++ source file makes the C++ language monolithic.

Furthermore I have the impression that you compare mostly with the original WEB (ie, Knuth's Pascal WEB). May you please include the capabilities of more modern WEBs (eg, FWEB, CWEB, noweb, or FunnelWEB) in your comparison as well? Otherwise I would not

consider it objective. And if you talk of a "WEB-environment": May you also consider some WEB environments as outlined in the usual bibligraphies on literate programming? There has been dissertations and a lot of publications in this area. Oh yes, and I don't understand why pseudo-comments are better than explicit tags. Both are markup.

Announcments for noweb

From: **Norman Ramsey**
Date: 18 Mar 1993

Although noweb has been available outside Princeton for more than two years, I am now officially announcing its release. What's "official" about this release is that I now have time to support noweb and I promise to fix bugs. noweb is available via anonymous ftp from bellcore.com in file ~ftp/pub/norman/noweb.shar.Z or from csservices.princeton.edu in file ~ftp/pub/noweb.shar.Z. DOS code with binaries is available in the same locations as dosnoweb.zip. I do NOT support the DOS code and it may or may not be the same as the supported source code. The rest of this announcement repeats information in the noweb README file; it contains a short sales pitch and a description of what you get and on what terms. I'm posting a longer article that contains a more in-depth description and sales pitch.

INTRODUCTION -- WHAT IS noweb?

noweb is designed to meet the needs of literate programmers while remaining as simple as possible. Its primary advantages are simplicity, extensibility, and language-independence. noweb uses 4 control sequences to WEB's 27, and its manual is only two pages. noweb works "out of the box" with any programming language, and its formatter-dependent part is under 50 lines. The primary sacrifice relative to WEB is the loss of the language-dependent features: prettyprinting and an index of identifiers. noweb provides extensibility by using the Unix toolkit philosophy. The "notangle" and "noweave" commands are built from pieces, which are then assembled in pipelines using shell scripts. The pieces are:

markup convert noweb file from human syntax to tool syntax
unmarkup inverse of markup
nt `tangle' the tool form of the noweb file
noxref insert cross-reference information for latex

These pieces are combined by the scripts in the shell directory to provide more than just weaving and tangling:

notangle analog of TANGLE
noweave analog of WEAVE
nountangle tangle, but keep interleaved documentation in comments
noroots print names of all root chunks in a noweb file
nocount count number of lines of code and documentation.

noweb has been used for three years both at Princeton and elsewhere. It has been used for tens of thousands of lines of code in such languages as awk, C, C++, Icon, Modula-3, Promela, and Standard ML. If you already know you want to use noweb, you need only install it and read the manual page. If you're just curious about noweb, a sales pitch appears in the technical report in doc/ieee.tex.

WHAT YOU GET IN THIS DISTRIBUTION

This distribution contains the following directories:
contrib software contributed by noweb users
doc man pages and a technical report
examples parts of noweb programs in different languages
icon Icon code for nonstandard weave and cross-referencer
lib noweave's cross-referencer
shell all the shell scripts that make up the actual commands
src source code for nt and markup
tex supporting tex code for /usr/local/lib/tex/macros
where appropriate, these directories have README files of their own.

WEAVING

The worst aspect of literate programming is the enormous amount of time wasted wrangling over what prettyprinted output should look like. Although noweb does no prettyprinting, it is not entirely immune--- several people have complained about noweave's output or have sent me changes that add more options to noweave. Having been down that road with Spider, I won't be fooled again. noweb doesn't try to be all things to all programmers, but it is very easy to change. If you don't like noweave's formatting, you can easily throw away noweave and make your own. To help you get started, the shell directory in the distribution contains three versions of noweave: noweave the standard (supposed to be latex-proof) noweave.nr what I use (handles 90 columns of code) noweave.simple simple, uses no special TeX hacking The simple version can't handle code with @ signs. The article in doc/ieee.tex explains the intermediate language that noweb uses to represent literate programs. noweb comes with two cross-referencers for use with noweave. The standard one is written in awk, because that's what everybody has. There is also a somewhat better cross-referencer written in Icon. Neither cross-referencer has been thoroughly exercised. See the INSTALL file for more details. noweb is designed to be extended with a language-dependent prettyprinter and indexer. I haven't written one because my experience with Spider taught me that prettyprinting is far more trouble than it's worth. If someone else wants to write one, I will be happy to help and advise.

NOTES

doc/ieee.* contains a paper that has been submitted to IEEE Software. You must `make install' before attempting to format the paper, since it uses the noweb document style option. The paper documents the representation of noweb files that is used by the noweb tools, in case you want to write any tools of your own. Simple tools (e.g. count the number of lines of interleaved documentation) are trivial. If you write any tools, or you want tools written (e.g. prettyprinters, index generators), let me know. The icon directory contains Icon programs that do most of the job of noweave.sh and noxref. If you want to adapt noweb to work with a text processor other than TeX or latex, they might provide a better starting point. I confess that the whole system should have been written in Icon from the beginning, but I'm not going to do it over. Icon is available by anonymous ftp from cs.arizona.edu. Thanks to Dave Hanson for cpif. Thanks to Joseph Reynolds for prodding me to fix [[...]]. Thanks to Lee Wittenberg for the DOS binaries. Thanks to Gary Leavens and Lee Wittenberg for testing this version, especially the installation process. I am, as always, responsible for errors and awkwardness that remain. Send comments or questions to norman@bellcore.com. I enjoy hearing from noweb users; if you have enjoyed noweb, why not send me a local postcard for my collection? My address is:

Norman Ramsey

Bellcore

445 South Street

Morristown, New Jersey 07960

USA

COPYRIGHT

From: **Norman Ramsey**
Date: 18 Mar 1993

This message contains an ASCII version of the technical report that describes noweb and why you might care to use it. The article may one day be published in IEEE Software (it's been under review for more than a year), but don't hold your breath. If you get noweb from csservices.princeton.edu, you get the TeX source for this article, and you can make a version you might actually be able to read.

Literate-Programming Tools Need Not Be Complex

Norman Ramsey

Department of Computer Science, Princeton University

35 Olden Street, Princeton, New Jersey 08544

August 1992

Abstract

When it was introduced, literate programming meant WEB. Desire to use WEB with languages other than Pascal led to the implementation of many versions. WEB is complex, and the difficulty of using WEB cre- ates an artificial barrier to experimentation with literate programming. noweb provides much of the functionality of WEB, with a fraction of the complexity. noweb is independent of the target programming language, and its formatter-dependent part is less than 40 lines. noweb is extensible, because it uses two representations of programs: one easily edited by authors and one easily manipulated by tools. This paper explains how to use the noweb tools and gives examples of their use. It sketches the implementation of the tools and describes how new tools are added to the set. Because WEB and noweb overlap, but each does some things that the other cannot, this paper enumerates the differences.

Key words: literate programming, readability, programming environments

Introduction

When literate programming was introduced, it was synonymous with WEB, a tool for writing literate Pascal programs [6, Chapter 4]. The idea attracted attention; several examples of literate programs were published, and a spe- cial forum was created to discuss literate programming [1, 2, 6, 13]. WEB was adapted to programming languages other than Pascal [3, 7, 8, 10, 12]. With experience, many WEB users became dissatisfied [9]. Some found WEB not worth the trouble, as did one author of the program appearing in Ap- pendix C of Reference 11. Others built their own systems for literate pro- gramming. The literate-programming forum was dropped, on the grounds that literate programming had become the province of those who could build their own tools [14].

WEB programmers interleave source code and descriptive text in a sin- gle document. When using WEB, a programmer divides the source code into modules. Each module has a documentation part and a code part, and mod- ules may be written in any order. The programmer is encouraged to choose an order that helps explain the program. The code parts are like macro definitions; they have names, and they contain both code and references to other modules. A WEB file represents a single program; TANGLE extracts that program from the WEB source. One special module has a code part with no name, and TANGLE expands the code part of that module to extract the program. WEAVE converts WEB source to TEX input, from which TEX can produce high-quality typeset documentation of the program.

WEB is a complex tool. In addition to enabling programmers to present pieces of a program in any order, it expands three kinds of macros, pret- typrints code, evaluates some constant expressions, provides an integer rep- resentation for string literals, and implements a simple form of version control. The manual for the original version documents 27 "control se- quences" [5]. The versions for languages other than Pascal offer slightly different functions and different sets of control sequences. Significant effort is required to make WEB usable with a new programming language, even when using a tool designed for that purpose [8].

WEB's shortcomings make it difficult to explore the idea of literate pro- gramming; too much effort is required to master the tool. I designed a new tool that is both simple and independent of the target programming language. noweb is designed around one idea: writing named chunks of code in any order, with interleaved documentation. Like WEB, and like all literate-programming tools, it can be used to write a program in pieces and to present those pieces in an order that helps explain the program. noweb's value lies in its simplicity, which shows that the idea of literate programming does not require the complexity of WEB.

noweb

A noweb file contains program source code interleaved with documentation. When notangle is given a noweb file, it writes the program on standard output. When noweave is given a noweb file, it reads the noweb source and produces, on standard output, TEX source for typeset documentation. Figure 1 shows how to use notangle and noweave to produce code and documentation for a C program contained in the noweb file foo.nw.

```
notangle foo.nw > foo.c
noweave foo.nw > foo.tex
```

Figure 1: Using noweb to build code and documentation

A noweb file is a sequence of chunks, which may appear in any order. A chunk may contain code or documentation. Documentation chunks begin with a line that starts with an at sign (@) followed by a space or newline. They have no names. Code chunks begin with <<chunk name>>= on a line by itself. The double left angle bracket (<<) must be in the first column. Chunks are terminated by the beginning of another chunk, or by end of file. If the first line in the file does not mark the beginning of a chunk, it is assumed to be the first line of a documentation chunk. Documentation chunks contain text that is ignored by notangle and copied verbatim to standard output by noweave (except for quoted code). noweave can work with LaTEX, or it can use a TEX macro package, supplied with noweb, that defines commands like "chapter" and "section".

Code chunks contain program source code and references to other code chunks. Several code chunks may have the same name; notangle concate- nates their definitions to produce a single chunk, just as TANGLE does. Code chunk definitions are like macro definitions; notangle extracts a program by expanding one chunk (by default the chunk named <<*>>). The defini- tion of that chunk contains references to other chunks, which are themselves expanded, and so on. notangle's output is readable; it preserves the inden- tation of expanded chunks with respect to the chunks in which they appear. Code may be quoted within documentation chunks by placing double square brackets around it ([[...]]). These double square brackets are ignored by notangle, but they are used by noweave to give code special typographic treatment. If double left and right angle brackets are not paired, they are treated as literal "<<" and ">>". Users can force any such brackets, even paired brackets, to be treated as literal by preceding the brackets by an at sign (e.g. "@<<").

```
@ This program has no input, because we want to keep it
simple. The result of the program will be to produce a
list of the first thousand prime numbers, and this list
will appear on the [[output]] file.
Since there is no input, we declare the value [[m = 1000]]
as a compile-time constant. The program itself is capable
of generating the first [[m]] prime numbers for any
```

```
positive [[m]], as long as the computer's finite
limitations are not exceeded.
<<program to print the first thousand prime numbers>>=
program print_primes(output);
const m = 1000;
<<other constants of the program>>
var <<variables of the program>>
begin <<print the first [[m]] prime numbers>>
end.
```

Figure 2: Sample noweb input, from prime number program

Figure 2 shows a fragment of a noweb program that computes prime numbers. The program is derived from the example used in Reference 6, Chapter 4, and Figure 2 should be compared with Figure 2b of that paper. Figure 3 shows the program after processing by noweave and LaTEX. Figure 4 shows the beginning of the program as extracted by notangle. A complete example program accompanies this paper.

```
This program has no input, because we want to keep it simple.
The result of the program will be to produce a list of the first
thousand prime numbers, and this list will appear on the output
file.
Since there is no input, we declare the value m = 1000 as a
compile-time constant. The program itself is capable of gen-
erating the first m prime numbers for any positive m, as long as
the computer's finite limitations are not exceeded.
<program to print the first thousand prime numbers>
program print_primes(output);
const m = 1000;
<other constants of the program>
var <variables of the program>
begin <print the first m prime numbers>
end.
```

Figure 3: Output produced by noweave and LaTEX from Figure 2

```
program print_primes(output);
const m = 1000;
rr = 50;
cc = 4;
ww = 10;
ord_max = 30; - p_ord_max squared must exceed p_m "
var p: array [1..m] of integer;
- the first m prime numbers, in increasing order "
page_number: integer;
..
.
```

Figure 4: Part of primes program as written by notangle

Using noweb

Experimenting with noweb is easy. noweb has little syntax: definition and use of code chunks, marking of documentation chunks, quoting of code, and quoting of brackets. noweb can be used with any programming language, and its manual fits on two pages. On a large project, it is essential that compilers and other tools be able to refer to locations in the noweb source, even though they work with notangle's output [9]. Giving notangle the -L option makes it emit prag- mas that inform compilers of the placement of lines in the noweb source. It also preserves the columns in which tokens appear. If notangle is not given the -L option, it respects the indentation

of its input, making its output easy to read. Large programs may also benefit from cross-reference infor- mation. If given the -x option, noweave uses LaTEX to show on what pages each chunk is defined and used.

WEB files map one to one with to both programs and documents. The mapping of noweb files to programs is many to many; the mapping of files to documents is many to one. Source files are combined by listing their names on notangle's or noweave's command line. Many programs may be extracted from one source by specifying the names of different root chunks, using notangle's -R command-line option. The simplest example of a one-to-many mapping of programs is that of putting C header and program in a single noweb file. The header comes from the root chunk <header>, and the program from the default root chunk, <*>. The following rules for make automate the process:

```
foo.c: foo.nw
notangle -L foo.nw > foo.c
foo.h: foo.nw
notangle -R header foo.nw > xfoo.h
-cmp -s xfoo.h foo.h __ cp xfoo.h foo.h
```

Using cmp avoids touching the header file when its contents haven't
changed. This trick is explained on pages 265-266 of Reference 4.

A more interesting example is using noweb to interleave different languages in one source file. I wrote an awk script that read a machine description and emitted a disassembler for that machine, and I used noweb to combine the script and description in a single file, so I could place each part of the input next to the code that processed that input. The machine description was in the root chunk <opcodes table>, and the awk script in the default root chunk. The processing steps were:

```
notangle opcodes.nw > opcodes.awk
notangle -R 'opcode table' opcodes.nw
awk -f opcodes.awk > disassem.sml
```

Many-to-one mapping of source to program can be used to obtain effects similar to those of Ada or Modula-3 generics. Figure 5 shows generic C code that supports lists. The code can be "instantiated" by combining it with another noweb file. pair_list.nw, shown in Figure 6, specifies lists of integer pairs. The two are combined by applying notangle to them both:

```
notangle pair_list.nw generic_list.nw > pair_list.c
```

noweb has no parameter mechanism, so the "generic" code must refer to a fixed set of symbols, and it cannot be checked for errors except by compiling pair_list.c. These restrictions make noweb a poor approximation to real generics, but useful nevertheless.

I have used noweb for small programs written in various languages, in- cluding C, Icon, awk, and Modula-3. Larger projects have included a code generator for Standard ML of New Jersey (written in Standard ML) and a multi-architecture debugger, written in Modula-3, C, and assembly lan- guage. A colleague used noweb to write an experimental file system in C++.

```
This list code supports circularly-linked lists represented by a pointer to
the last element. It is intended to be combined with other noweb code that
defines <fields of a list element> (the fields found in an element of a list) and
that uses <list declarations> and <list definitions>.
<list declarations>
typedef struct list -
<fields of a list element>
struct list *_link;
" *List;
extern List singleton(void); /* singleton list, uninitialized fields */
extern List append(List, List); /* destructively append two lists */
#define last(l) (l)
#define head(l) ((l) ? (l)->next : 0)
#define forlist(p,l) for (p=head(l); p; p=(p==last(l) ? 0 : p->next))
<list definitions>
List append (List left, List right) -
```

```
List temp;
if (left == 0) return right;
if (right == 0) return left;
temp = left->_link; left->_link = right->_link; right->_link = temp;
return right;
"
.
```

Figure 5: Generic code for implementing lists in C

```
<*>
<list declarations>
<list definitions>
<fields of a list element>
int x;
int y;
```

Figure 6: Program to instantiate lists of integer pairs

The sizes of these programs are

Program	Documentation lines	Total lines
markup and nt	400	1,200
ML code generator	900	2,600
Debugger	1,400	11,000
File system	4,400	27,000

Representation of noweb files

The noweb syntax is easy to read, write, and edit, but it is not easily manipu- lated by programs. To make it easy to extend noweb, I have written markup, which converts noweb source to a representation that is easily manipulated by commonly used Unix tools like sed and awk. In this representation, every line begins with @ and a key word. The possibilities are:

@begin kind n Start a chunk
@end kind n End a chunk
@text string string appeared in a chunk
@nl A newline
@defn name The code chunk named name is being defined
@use name A reference to code chunk named name
@quote Start of quoted code in a documentation chunk
@endquote End of quoted code in a documentation chunk
@file filename Name of the file from which the chunks came
@literal text noweave copies text to output

markup numbers each chunk, starting at 0. It also recognizes and undoes the escape sequence for double brackets, e.g. converting "@<<" to "<<". markup's output represents a sequence of files. Each file is represented by a "@file filename" line, followed by a sequence of chunks. The representation of a documentation chunk where docline may be @text, @nl, @quote, or @endquote is:

@begin docs n where n is the chunk number.
docline repeated an arbitrary number of times.
@end docs n

Every @nlcorre-sponds to a newline in the original file. markup guarantees that quotes are balanced and not nested. The representation of a code chunk where codeline may be @text, @nl, or @use is:

```
@begin code n where n is the chunk number.
@defn name    name of this chunk.
@nl           The newline following <<name>>= in the original file
codeline      repeated an arbitrary number of times.
```

```
@end code n
```

The noweb tools are implemented by piping the output of markup to other programs. notangle is a Unix shell script that builds a pipeline be- tween markup and nt, which reads and expands definitions of code chunks. noweave pipes the output of markup to a 24-line awk script that inserts appropriate TEX or LaTEX formatting commands. Having a format easily read by programs makes noweb extensible; one can manipulate literate programs using Unix shell scripts and filters. To be able to share programs with colleagues who don't enjoy literate program- ming, I modified notangle by adding to its pipeline a stage that places each line of documentation in a comment and moves it to the succeeding code chunk. The resulting script, nountangle, transforms a literate program into a traditional commented program, without loss of information and with only a modest penalty in readability. Figure 7 shows the results of applying nountangle to the prime-number program shown in Figure 2. noweave's cross-reference generation is also implemented as an extension; the output of markup is piped through an awk script that uses @literal to insert LaTEX cross-reference commands. Another simple tool finds all the roots in a noweb file, making it easy to find definitions where chunk names have been misspelled.

Comparing WEB and noweb

Unlike WEB, noweb is independent of the target programming language. WEB tools can be generated for many programming languages, but those lan- guages must be lexically similar to C. For example, WEB can't handle the awk regular-expression notation "/:::/"; every such expression must quoted using WEB's "verbatim" control sequence. The effort required to generate WEB tools is significant; the prospective user must write a specification of several hundred lines.

```
- This program has no input, because we want to keep it   "
- simple. The result of the program will be to produce a "
- list of the first thousand prime numbers, and this list "
- will appear on the [[output]] file. "
..
.
- <program to print the first thousand prime numbers>= "
program print_primes(output);
const m = 1000;
- "section-The output phase- "
- "
- <other constants of the program>= "
rr = 50;
cc = 4;
ww = 10;
- <other constants of the program>= "
ord_max = 30; - p_ord_max squared must exceed p_m "
var - How should table [[p]] be represented? Two possibilities "
- suggest themselves: We could construct a sufficiently "
..
.
```

Figure 7: Output produced by nountangle from Figure 2

Being independent of the target programming language makes noweb simpler, but it also means that noweb can do less. Most of the differences between WEB and noweb arise because WEB has language-dependent features that are not present in noweb. These features include prettyprinting, type- setting comments using TEX, generating an index of identifiers, expanding macros, evaluating constant expressions, and converting string literals to indices into a "string pool." Among these features, noweb users are most likely to miss prettyprinting and the index of identifiers. Some differences arise because WEB and noweb implement similar features differently. WEB's original TANGLE removed white space and folded lines to fill each line with tokens, making its output unreadable [6, Chapter 4, Figure 3]. Later adaptations preserved line breaks but removed other white space. By default, notangle preserves whitespace and maintains indentation when expanding chunks. It can therefore be used with languages like Miranda and Haskell, in which indentation is significant. TANGLE cannot. WEB's WEAVE assigns a number to each chunk, and its cross-reference information refers to chunk numbers, not page numbers. noweb uses LaTEX to emit cross-reference information that refers to page numbers. Anyone who has read a large literate program will appreciate the difference.

WEB works poorly with LaTEX; LaTEX constructs cannot be used in WEB source, and getting WEAVE output to work in LaTEX documents requires te- dious adjustments by hand. noweb works with both plain TEX and LaTEX. Both WEAVE and noweave depend on the text formatter in two ways: the source of the program itself, and the supporting macros. WEAVE's source (written using WEB for C) is several thousand lines long, and the formatting code is not isolated. noweave's source is a 57-line shell script, and only 31 of

those lines have to do with formatting. Both WEAVE and noweave use about 200 lines of supporting macros for plain TEX. noweb uses another 80 lines to support LaTEX, most of which is used to eliminate duplicate page numbers in cross-reference lists.

noweb has two features that weren't in the original WEB, but that appeared in some of WEB's later adaptations. They are the ability to inform the compiler of the original locations of source lines and the ability to extract more than one program from a single source file. Reviewers have had many expectations of literate-programming tools [13, 14]. The most important is verisimilitude: a single input should produce both compilable program and publishable document, warranting the correctness of the document. Others include flexible order of elaboration, ability to develop program and documentation concurrently in one place, cross-references, and indexing. WEB satisfies all these expectations, and noweb satisfies all but one (it does not provide automatic indexing).

Discussion

WEB takes the monolithic approach to literate programmingit does everything. noweb's approach is to compose simple tools that manipulate files in the noweb format. Existing Unix tools provide some of the WEB features that aren't found in noweb. Unix supplies two macro processors: the C preprocessor and the m4 macro processor. xstr extracts string literals. patch provides a form of version control similar to WEB's change files. Few of WEB's remaining features will be missed; for example, many compilers evaluate constant expressions at compile time. Experience with WEB has suggested that prettyprinting may be more trouble than it is worth, and that the index of identifiers, while useful, is not a necessity [9]. Three things distinguish noweb from previous work. noweb takes as simple as possible a view of literate programming and the tools needed to implement it. Instead of relying on a generator or re-implementation to support different programming languages, noweb is independent of the target programming language. noweave's dependence on its typesetter is small and isolated, instead of being distributed throughout a large implementation. Experimenting with noweb is easy because the tools are simple and they work with any language. If the experiment is unsatisfying, it is easy to abandon, because notangle's output, unlike TANGLE's, is readable. noweb is simpler than WEB and is easier to use and understand, but it does less. I argue, however, that the benefit of WEB's extra features is outweighed by cost of the extra complexity, making noweb better for writing literate programs. noweb can be obtained by anonymous ftp from csservices.princeton.edu, in file pub/noweb.shar.Z.

Acknowledgements

Mark Weiser's invaluable encouragement provided the impetus for me to write this paper, which I did while visiting the Computer Science Labo- ratory of the Xerox Palo Alto Research Center. Comments from David Hanson and from the anonymous referees stimulated me to improve the pa- per. The development of noweb was supported by a Fannie and John Hertz Foundation Fellowship.

References

[1] P. J. Denning. Announcing literate programming. Communications of the ACM, 30(7):593, July 1987.

[2] D. Gries and J. Bentley. Programming pearls: Abstract data types. Communications of the ACM, 30(4):284-290, April 1987.

[3] K. Guntermann and J. Schrod. WEB adapted to C. TUGboat, 7(3):134- 137, October 1986.

[4] B. W. Kernighan and R. Pike. The UNIX Programming Environment. Prentice-Hall, 1984.

[5] D. E. Knuth. The WEB system of structured documentation. Technical Report 980, Stanford Computer Science, Stanford, California, September 1983.

[6] D. E. Knuth. Literate Programming, volume 27 of Center for the Study of Language and Information Lecture Notes. Leland Stanford Junior University, Stanford, California, 1992.

[7] S. Levy. WEB adapted to C, another approach. TUGBoat, 8(1):12-13, 1987.

[8] N. Ramsey. Literate programming: Weaving a language-independent WEB. Communications of the ACM, 32(9):1051-1055, September 1989.

[9] N. Ramsey and C. Marceau. Literate programming on a team project. Software Practice & Experience, 21(7):677-683, July 1991.

[10] W. Sewell. How to MANGLE your software: the WEB system for Modula-2. TUGboat, 8(2):118-128, July 1987.

[11] W. Sewell. Weaving a Program: Literate Programming in WEB. Van Nostrand Reinhold, New York, 1989.

[12] H. Thimbleby. Experiences of `literate programming' using CWEB (a variant of Knuth's WEB). Computer Journal, 29(3):201-211, 1986.

[13] H. Thimbleby. A review of Donald C. Lindsay's text file difference utility, diff. Communications of the ACM, 32(6):752-755, June 1989.

[14] C. J. Van Wyk. Literate programming: An assessment. Communications of the ACM, 33(3):361-365, March 1990.

Announcment of WinWord WEB

From: **Lee Wittenberg**
Date: 26 Mar 1993

I have just completed a crude (repeat crude) WEB system that works with Microsoft Word for Windows (v2.0). Anyone who wants to play with it is welcome to get the file pub/leew/wordweb.zip via anonymous ftp to bart.kean.edu. The distribution includes a Word template with special macros, menu items, and a paragraph style defined (WORDWEB.DOT), a mercifully brief description of the system and how to use it (WORDWEB.DOC), and a trivial example program (EXAMPLE1.DOC).

This distribution is really a case of what you see is all you get. I do not intend on supporting this product (I don't use it myself; I'm still an unrepentant TeXXie), but there was such an extended discussion on the network about such an animal that it seems worthwhile to make my efforts generally available. If anyone is interested in supporting WinWordWEB on a regular basis, please let me know.

Ironically, the reason I ended up "belling the cat" myself (and being hoist by my own petard) is due to a colleague's success using noweb. I'd finally convinced one of my coworkers here to try literate programming. Once he got used to it, he was so ecstatic that everyone else was interested in trying it, but no one wanted to learn a new tool. WinWordWEB is the result. I'll be very interested in the experiences of anyone who tries using the package, particularly if you're interested in taking over this project. The program has a number of flaws, but is definitely usable. I will respond to all bug reports, but I don't promise to fix any of them.

Announcment of nuweb

From: **Preston Briggs**
Date: 07 Apr 1993

Inspired by all these nice new tools, especially FunnelWeb and noweb, I've written yet another web-like system. Currently I call it nuweb, punning slighly on Ramsey's noweb. It's similar to FunnelWeb in that it allows use of many programming languages and allowing the creation of many output files. On the other hand, it is much simpler and depends on latex (versus tex). I also stole the idea of writing output files to temps first and comparing before touching the final result. This is a _really_ nice feature for large programs; everyone should do it.

The big advantages I see for latex are the multilevel sectioning commands and the support for pictures (and bibliographies, crossreferences, etc). On the other hand, latex does add some overhead to programming. It may make nuweb better for writing papers and large programs than for smaller examples. Basically, you write a latex document, being slightly careful about using @ characters, but otherwise using all the features of latex. You can sprinkle in file definitions and macro definitions throughout the code. These'll be recognized and formatted slightly (code just comes out it \tt, with the no changes in line breaks or indentation).

There's only one program. You run it on the web file and it spits out the .tex file and all the output files at once. Basically, it's so fast, I don't see a lot of reason not to do everything at once. On the other hand, I defined flags to suppress different passes for use on slower machines. When the tex file is written, the file and macro definitions get cross reference info similar that provided by weave. I can make indices for file names and macro names, but I can't make an index of identifiers.

Thus, the big losses compared to a language-sensitive system like CWEB are: no pretty printing, no identifier index, and no line number info inserted in the output files. The big advantages are: no pretty printing, speed, and precise control of the output. It can be used it on any language or combination of languages (C, C++, Fortran, yacc, lex, awk, make, and so forth).

There aren't very many commands:
@@ puts a single @ into the result (tex or output file)
@i file_name includes a file
@o file-name scrap writes the scrap out to the file
@d macro-name scrap defines a macro that can be invoked from within a scrap
where scrap is
@{anything@} where "anything" doesn't include @o, @d, @m, @f, or @{
but may include @<macro-name@>
Every character between the @{ and the @} is significant, so be
careful of those carriage returns. In the .tex output, a $\Diamond$
is inserted to help show how many newlines are at the end of a scrap.
@f makes an index of file names
@m makes an index of macro names
@<macro-name@> may appear in a scrap. On output to a file,
will be expanded to include all the scraps
making up a macro. The expansion will be
indented to match the invocation.
A file or a macro may be defined in several places. All the scraps
contributing to their definitions are simply concatenated.
I allow ... at the end of macro names (a la Knuth), so you can type
@d This...
as an abbreviation for

@d This is a long macro name

I wrote it all in a hurry over a weekend. Then I rewrote it in nuweb. This took a little longer, since I was being more careful (and fixing features that had turned out poorly). I'm still filling in documentation -- always hard for me. I'll be glad to share it around if people will send me bug reports. The code is all in C (old-style, not ANSI C). Seems to work on a Sun; ought to work on other things. Doesn't require awk or anything; but it doesn't make a lot of sense without latex.

Announcements for FWEB

From: **John Krommes**
Date: 14 Jun 1993

Version 1.30 (non-beta version) of FWEB is now available from "lyman.pppl.gov:/pub/fweb/fweb-1.30.tar.Z". For a short list of things that have changed or been fixed between fweb-1.30--beta and fweb-1.30, see /pub/fweb/READ_ME.

To obtain, use anonymous guest ftp in binary mode to obtain the tar file, then: "uncompress fweb-1.30.tar" and "tar -xvf fweb-1.30.tar". This creates the directory fweb-1.30. Note that the installation procedure has changed for Unix users, as I slowly gravitate toward gnu-like conventions. FWEB now attempts to configure itself automatically to the host environment:

cd fweb-1.30

./configure

cd web

make bootstrap

make -n install (Use the -n option to see where various files are going to be put. If you need to make changes, make them in defaults.mk.in, then rerun ./configure.)

[Log on as root]

make install

WHO SHOULD USE FWEB?

FWEB is a language-sensitive WEB originally based on CWEB. It was initially intended to support Fortran (hence the 'F'); however, an important feature of FWEB is its ability to support particular multiple languages. The significant features of FWEB are:

- multiple language support (even within the same WEB run): C, C++, Fortran--77, Fortran--90, Ratfor, and (to a limited extent) TeX;
- built-in Ratfor translator (into either Fortran--77 or Fortran--90);
- built-in ANSI-C-like macro preprocessor (with various special-purpose extensions);
- superior cross-referencing facilities for identifiers;
- sophisticated facilities for changing the typeset appearance of operators and identifiers (these work either with the operator-overloading features of C++ and Fortran--90, or stand-alone);
- extensive customization facilities via a style file;
- usable with either TeX or LaTeX;
- has been installed successfully on a great variety of machines, including everything from IBM-PCs to CRAYs.

You should use FWEB if:

- you're a Fortran programmer;
- you mix languages;
- you need very flexible macro-processing facilities;
- you enjoy a variety of convenience features.

You should (perhaps) not use FWEB if:

- you like language-insensitive literate programming;
- you need a very small and lean system (FWEB's internal tables are larger than those of some other WEB's because of the multiple language support and various advanced cross-referencing facilities; the macro processor and convenience features enlarge it still more);
- you program only in C or C++ and are perfectly happy with CWEB.

DOCUMENTATION

If you're an experienced literate programmer and want to find out whether FWEB is for you, tex and read the relatively short reference/syntax guide in fweb-1.30/manual/guide: "cd manual" and "make guide". FWEB also features a very large and complete user manual suitable for beginners; this contains many examples and an extensive index and table of contents: "cd manual" and "make manual".

SUPPORT

FWEB remains actively supported, although STRICTLY AS A SPARE-TIME ACTIVITY. It is available courtesy of Princeton University's Plasma Physics Laboratory, a contractor of the U.S. Dept. of Energy whose mission is research into fusion energy. The nation fusion program is currently under great stress, and this affects the feasible level of support. Questions, bug reports, and suggestions of all kinds are still very welcome; however, I can no longer guarantee immediate response. I will attempt to answer questions about installation quickly, and will provide workarounds for egregious bugs. Suggestions for new features are filed away systematically and are reviewed periodically.

Announcements for CWEBx

From: **Marc van Leeuwen**
Date: 16 Dec 1993

Announcement of CWEB 3.x, a new and modified version of CWEB. A new version of CWEB, called version 3.x, has been developed and is now officially being released. It is available by anonymous ftp from ftp.cwi.nl, in the directory pub/cweb. The version was developed from the original CWEB of Levy/Knuth, version 2.1, via an intermediate version called auc-cweb by Frank Jensen of Aalborg University. The following list gives the differences with respect to the original CWEB version; these are minor but useful ones, of which the more significant ones were deemed necessary in order to allow existing moderately large C programs to be gradually adapted to a literate style without requiring major organisational changes.

- The programs are fully adapted to ANSI C: they support literate programs in that language and are themselves written in ANSI C.
- The CWEB system deals explicitly with #include preprocessor directives, just like #define directives were already handled. There are two important reasons for this: firstly the programmer is given control over the relative order of #include and #define directives in the output of CTANGLE, so that the situation that macro definitions disrupt the reading of (system) header files can be avoided, and secondly it allows CWEAVE to be aware of typedef declarations that are hidden in header files, so that their identifiers can be treated properly.
- Typedef declarations are found during the first pass of CWEAVE, along with all other |ilk| assingments. This implies that code containing typedef identifiers will be formatted correctly, whether it precedes the typedef declaration or follows it; this conforms to the WEB philosophy of giving the programmer maximal freedom in ordering the material, and also avoids awkward problems when discussing a typedef declaration in its own commentary (apart from the move to ANSI C, this is the change that required the most significant change of the program sources).
- A mechanism is provided to explicitly refer to the number of specific sections from within the commentary text, by means of symbolic labels.
- Module names are treated syntactically as statements by default, rather than as expressions (although means are provided to treat them as declarations or as expressions). This is really removing a property that was unjustly retained from (Pascal) WEB, since in C statements and expressions have completely separate places in the syntax, and module names almost never stand for expressions; the practical consequence is that it removes the need of putting an invisible semicolon (@;) after almost every module name (or even worse a real semicolon).
- A few (further) control codes have been added for the following purposes: to introduce a new section without risking a page break before it (useful between modules that are closely related but have to be separate because they are CTANGLEd to different places); to use the `|' (bitwise or) operator within `|...|'; and to remove one level of indentation from a line (useful for module names that stand for cases in a switch statement).
- Arbitrary 8-bit characters may be used in source files. The programs are completely robust against the use of such characters, but using them in program fragments outside strings and comments causes an error.
- Several styles of layout can be selected by command line options. The default style aligns all matching braces vertically (unlike other versions of CWEB) without consuming extra space.
- CWEAVE can be told to report irreducible scrap sequences (which cause bad layout) by a command line option, i.e., without changing the source file.
- The command line syntax has been changed so that with standard naming conventions only a single file name argument is needed, whether or not a change file is being used.
- Breaking of long expressions across lines, if necessary, is done in an intelligent way, favouring a break at operators of lowest priority, except when enclosed in parentheses or square brackets. (This is really only a change of the cwebmac format used by TeX.)

- A completely new manual has been written (25 pages) providing a (hopefully) more didactical introduction to literate programming and the use of CWEB; it contains a simple but complete sample program illustrating all basic aspects of CWEB, and also a one-page table of all control codes.

Furthermore significant changes have been made to the program sources; although these are of little concern to ordinary users, they should make it easier to maintain the programs or to modify them to handle different but similar tasks (e.g., handle a different programming language); indeed they were intended to make the programs more "literate". Although the basic structure of the programs has not been greatly altered, almost every part of the source files has undergone rewriting; many minor errors were removed in this process. Some particularly noteworthy points:

- All trivial system dependencies have been removed, so that generally no change file is necessary to compile CWEB itself (this mainly means that the referral to "/dev/null" has been eliminated); the only explicit system dependencies are the assumption of the ASCII character set and the assumed interpretation of the value returned from |main|.
- The parser of CWEAVE operates by interpretation of a set of grammar rules that are explicitly stored in a (statically initialised linear) table. The formatting rules are given by easily interpreted format strings. Altogether there is a clean separation between parser and grammar (somewhat along the lines of the SPIDER system), making it relatively easy to alter the set of rules; in fact a system for dynamically selecting rules according to command line arguments is present that could easily be extended to support more variants.
- The programs use a header file read in by #include to share common declarations between different compilation units (as is normal practice in multi-file C programs), instead of using the CWEB inclusion facility (@i) for this, which would lead to the text being replicated into all CWEB documents. In doing so, the programs also set a good example of how the multiple output file facility can be effectively used, since the header file is produced by CTANGLE from the same source file (common.w) that contains the definitions of the objects that the declarations in the header file refer to.
- The style of the programs has been altered to a more C-like one on numerous points where there were complications that could only be explained by the Pascal origins of the programs. For instance, since the program usually manipulates pointers into arrays rather than the indices of the elements, it is more natural to use NULL to indicate an exceptional value (e.g., the end of a list) than the value of a pointer to a specially reserved element.
- In programs and documentation a systematic distinction is made between sections, which are numbered pieces of the CWEB document that may or may not contain a program fragment, and modules, which are named pieces of C code that are defined in one or more sections; thus a source of confusion is removed.

As I recently found out, there is now an updated version of the original Levy/Knuth CWEB, called version 3.1. Therefore the version announced here introduces a forking in the development history of CWEB, which is of course regrettable, and which explains the strange version number of this version, that would otherwise have been called CWEB 3.0. Apparently the main change in version 3.1 is the support for C++ and therefore a fortiori for ANSI C; further small changes have been made in direction different from those for version 3.x, such as handling 8-bit characters in identifiers. Since it is my intention to serve the literate programming community, not to divide it, I will make an attempt at some future time to merge the two branches by incorporating those additions if version 3.1 that are still relevant to 3.x, most importantly to extend the support for C++ (currently the support is limited to what was present in version 2.1, which is not likely to be sufficient in view of the evolutionary nature of C++). Meanwhile I deemed it wise not to hold up the release of the current version until such changes have been made.

In other respects as well, I plan to provide active support for CWEB 3.x. For one thing, it will be used to convert the computer algebra program LiE, which is moderately large (currently about 600Kb source code in some 65 files), to a literate program; this should provide a good test for the practical usability. Therefore all bug reports, comments and suggested can be sent to the address below.
