CLiP, a universal literate programming tool

Eric. W. 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

Remark: A postscript edition of this text can be anonymously FTP-ed from

ftp_node:	sun01.info.wau.nl (IPnr 137.224.133.1)
directory:	clip
file:	clip_ann.ps

1 Introduction

It is clearly impossible to realise literate programming (LP) without a supporting tool. Historically Knuth was the first to report the very idea of LP using his WEB-system and most LP 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 LP-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 (LP-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 LP-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) inputfiles 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 sourcefiles.
- 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 LITPROG 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 LP 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.

I N F O R M A T I O N

For any information on the CLiP-system please contact

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