```
/*
```

Introduction.

This program prints factorials of positive integers. It uses this definition of $n!$

$$n! = \prod_{i=1}^{n} i$$

⟦ **pf.c** begins with a comment which is treated as TEX-text by c-~~web~~.

The first comment contains the string `\input cnoweb`. This begins the c-~~web~~ listing. Included also in this leading comment section is `\title{Printing ...}`, `\synopsis{You are reading ...}`, and `\section{Introduction}`. ⟧

The program is invoked with the command

pf [**-l**] [**-e**] n

where

-l prints the factorials of all numbers up to and including the limit n.

-e includes an exponential notation approximation.

n is the number whose factorial will be printed. `*/`

```
#define GOODEXIT 0          /* exit with this if factorial OK */
#define BADEXIT  1          /* exit with this if error */

#include <stdio.h>          /* standard io stuff */
#include <math.h>           /* math stuff */
char     *malloc();         /* memory allocator */
```

/* Big-integer routines.

pf would be trivial were it not for the rapid growth of $n!$, which is much greater than exponential and quickly surpasses even the real number capacity of most machines. We therefore define a new data type (**Int**) to hold big integers. `*/`

```
typedef struct Int_ {
   int radix;                /* 2^1 ≤ radix ≤ 2^8 */
   char *digits;             /* the Int's digits */
   int length;               /* number of digits available */
   int limit;                /* number of digits in use */
} Int;
```

```
/* We do not need many operations on Ints to compute factorials: only 'set' and 'multiply'. */
```

```
/* Set(I, i). Return I initialized to i. */

Set(I,i)
Int *I;
int i;
{
   int d;
   I->digits[0] = (i?1:0);
   for (d=1; d<I->length; I->digits[d++]=0);
   I->limit = 0;
   if ((i!=0)&&(i!=1)) Product(I,i);
   return (0);
}

/* Product(I, i). Return I multiplied by i. Returns true if the product is OK. */

Product(I,i)
Int *I;
int i;
{
   int d,n;
   int l = I->limit+1;
   int r = I->radix;
   int carry = 0;

   for (d=0; ((d<l)||(carry!=0)); d++) {
      if (d>=I->length) return (0);       /* not enough digits */
      n = I->digits[d] * i + carry;
      if ((I->digits[d] = n % r)>0) I->limit = d;
      carry = n / r;
   }
   return (1);
}
```

```
/* main(). Print the factorial of the integer found on the command line.
```

1. Parse command line arguments.
2. Compute number of digits needed and allocate space.
3. Calculate the factorial.
4. Print the factorial.

⟦ c-~~web~~ breaks pages only before comments.
You can control pagination with this knowledge. ⟧

The higher the radix the fewer digits needed, but we use an internal radix of 10 to save a lot of division when printing the number. Also, we already have a $\log_{10}$ function.

```
*/

#define RADIX 10

Int nfactorial;       /* place for the factorial */
Int *nf = &nfactorial;


/* command line parameters */

int limit_mode = 0;         /* true if printing all factorials */
int expon_mode = 0;         /* true if printing exponential notation */


/* start of the program */

main(argc,argv)
int  argc;
char *argv[];
{
   int n;
   int i;


   /* 1. Parse the command line */

   n = parse_args(argc,argv);  /* we will print n! */


   /* 2. Compute digits needed and allocate space */

   nf->radix = RADIX;
   nf->length = digits(n,RADIX);

   if ((nf->digits = malloc(nf->length))==NULL) {
      printf("Sorry, cannot allocate space for %d digits\n",
           nf->length);
      exit (BADEXIT);
   }
```

```
   /* 3. Calculate the factorial. */

   Set(nf,1);          /* initialize nf */

   for (i=1; i<=n; i++) {
      if (Product(nf,i)==0) {
         printf("Sorry, miscalculated the number of digits.\n");
         exit (BADEXIT);
      }
      if (limit_mode) print_Int(nf,i);
   }


   /* 4. Print the result, if it hasn't been printed yet. */

   if (!limit_mode) print_Int(nf,n);
   exit (GOODEXIT);
}
```

/* **parse_args(argc,argv)**.

This is a standard UNIX idiom. See Kernighan & Ritchie.

⟦ c-~~web~~ automatically indents after `{` and `(`. ⟧ */

```
parse_args(argc,argv)
int argc;
char *argv[];
{
   int n = (-1);

   while (--argc > 0) {
      argv++;
      if (argv[0][0]=='-') {    /* is an option flag */
         switch (argv[0][1]) {
            case 'l': {
               limit_mode = 1;
               break;
            }
            case 'e': {
               expon_mode = 1;
               break;
            }
            default: show_usage();
         }
      } else {                  /* is the number */
         sscanf(argv[0],"%d",&n);
      }
   }

   if (n<=0) show_usage();
   return (n);
}
```

`/* digits(n,r)`. Returns number of digits in **n** using a radix of **r**.

A factorial can be approximated by the Sterling formula

$$n! \approx e^{-n} n^n \sqrt{2\pi n}$$

What we want is the number of digits, which is

$$\#\ \text{digits} \approx \log_r(n!) = \log(n!)/\log(r) \approx (-n\log e + n\log n + \frac{1}{2}\log(2\pi n))/\log(r)$$

We will add a couple of digits to this value to allow for the approximations and assure that we have enough. `*/`

```
#define PI    3.14159265358979323846264338   /* π */
#define E     2.71828182845904523536028747   /* e */

digits(n,r)
int n;
int r;
{
   double dn = n;
   int nd;

   nd = (int)((-dn * log10(E)) +
        (dn * log10(dn)) + (.5 * log10(2*PI*dn)))/log10((double)r) + 2;

   /* printf("requires %d digits\n",nd); */

   /* ⟦ The 'Commented out' code above was typed: /* *<printf...;>* */. ⟧ */

   return (nd);
}
```

/* **print_Int(I, i)**. Print *value(*`i`*) = value(*`I`*)*. The internal radix that matches the display radix clearly makes this procedure easier.

This display uses the convention that separates each three-digit set with commas. */

```
#define MAX_WIDTH 80              /* maximum width of the output */

print_Int(I,i)
Int *I;
int i;
{
   int d;
   int lp,bp;
   char line[MAX_WIDTH];

   sprintf(line,"%d! = ",i);
   bp = lp = strlen(line);
   for (d=I->limit; d>=0; d--) {
      line[lp++] = I->digits[d] + '0';  /* assumes radix ≤ 10 */
      if ((d%3)==0) {
         line[lp++] = (d?',':'.');
         if ((d==0) || (lp>MAX_WIDTH-5)) {
            line[lp] = '\0';
            printf("%s\n",line);
            for (lp=0; lp<bp; line[lp++]=' ');
         }
      }
   }


   /* If the exponential notation was desired, print it now. */

   if (expon_mode) {
      int d = I->limit;        /* the most significant digit */
      sprintf(line+bp,"approximately = %d.%d x 10e%d",
           I->digits[d],d?I->digits[d-1]:0,d);
      printf("%s\n",line);
   }
   return (0);
}


/* show_usage(). invocation syntax error—show correct usage */

int show_usage()
{
   printf("usage: pf [-l] [-e] positive_integer\n");
   exit (BADEXIT);
}
```

/* Summary of c-~~web~~ commands.

These control sequences are defined in the c-~~web~~ macro package.

\title{ ... }	Titles the program.
\job{ ... }	Another title area. Defaults to input filename.
\section{ ... }	Begins a section. The section title is also included in the table of contents and in the page header.
\subsection{ ... }	Begins a subsection. The subsection title is also included in the table of contents.
\subsubsection{ ... }	Begins a subsubsection. The subsubsection title is also included in the table of contents.
\newpage	Causes a page eject after the current line. This is usually used in a comment by itself, e.g., `/* \newpage */`.
\endc	Ends the c-~~web~~ listing. This is usually the last line in the file, e.g., `/* \endc */`.
\" **...** "	Prints **bold** text.
\' **...** '	Prints *italic* text.
| **...** |	Prints `typewriter` text.

*< **...** >*	Allows C code to be included in comments. You can nest comments within the 'commented out' C code, e.g.,
	`/* comment out this section *<`
	`  i = 0; /* initialize i */`
	`  ...`
	`  >* end of the commented out section */`

\item, \hang, etc.	Work as you hope they would.

*/

/* How to obtain c-~~web~~.

You may obtain c-~~web~~ by anonymous ftp to `u.washington.edu`.

It is in the directory: `pub/tex/cnoweb`

— Jim Fox, University of Washington
`fox@cac.washington.edu`

The file ends with the comment '`/* \endc */`' */

/* *end* */

pf

Printing factorials: a demonstration of c-~~web~~

You are reading a program listing that was formatted with

```
% tex pf.c
```

The same program is compiled with

```
% cc pf.c
```

The key to this dual function source file is a TEX macro package called c-~~web~~ that treats all comments as 'TEX-text' and all else as 'verbatim-text.' The C compiler naturally does the opposite and interprets only the text outside the comments.

⟦ In this program comments about c-~~web~~ itself look like this. ⟧

Page

Introduction 1
Big-integer routines 1
Set(I, i) 2
Product(I, i) 2
main() 3
parse_args(argc,argv) 5
digits(n,r) 6
print_Int(I, i) 7
show_usage() 7
Summary of c-~~web~~ commands 8
How to obtain c-~~web~~ 8