(A new global variable, *trace_depth*, is declared somewhere and initialized to zero. It is used to indent the output of DemoTEX so that the depth of subroutine nesting is displayed.)

At the beginning of *expand* (in §366), we put the statements

```
incr(trace_depth);
if tracing_stats > 2 then print("␣<x");
```

this prints '<x' when *expand* begins to expand something. The same statements are inserted at the beginning of *scan_int* (§400), *scan_dimen* (§448), and *scan_glue* (sec461), except that *scan_int* prints '<i', *scan_dimen* prints '<d', and *scan_glue* prints '<g'. (Get it?) We also insert complementary code at the end of each of these procedures:

```
decr(trace_depth);
if tracing_stats > 2 then print_char(">");
```

this makes it clear when each part of the scanner has done its work.

Finally, *scan_keyword* is instrumented in a similar way, but with explicit information about what keyword it is seeking. The code

```
incr(trace_depth);
if tracing_stats > 2 then
  begin print("␣<´"); print(s);
  print_char("´");
  end;
```

is inserted at the beginning of §407, and

```
  if tracing_stats > 2 then print_char("*");
exit: decr(trace_depth);
  if tracing_stats > 2 then print_char(">");
  end;
```

replaces the code at the end. (Here '*' denotes 'success': the keyword was found.)

For example, here's the beginning of what DemoTEX prints out when scanning the right-hand side of the assignment to `\hfuzz` in problem 18:

```
|! the character = <d
 |! the character 1 <i
  |! the character 1
  |! the character 0
  |! the character 0
  |! the letter P>
 |! the letter P <'em'
  |! the letter P> <'ex'
  |! the letter P> <'true'
  |! the letter P> <'pt'
  |! the letter P
  |! \ifdim =\ifdim <x <d
    |! the character 1 <i
     |! the character 1
     |! the character 2
      |! the letter p>
     |! the letter p <'em'
      |! the letter p> <'ex'
      |! the letter p> <'true'
      |! the letter p> <'pt'
      |! the letter p
      |! the letter t*>
     |! the character =>
```

(After seeing '=', TEX calls *scan_dimen*. The next character seen is '1'; *scan_dimen* puts it back to be read again and calls *scan_int*, which finds '100', etc. This output demonstrates the fact that TEX frequently uses *back_input* to reread a character, when it isn't quite ready to deal with that character.)

Acknowledgement

I wish to thank the brave students of my experimental class for motivating me to think of these questions, for sticking with me when the questions were impossible to understand, and for making many improvements to my original answers.

◇ Donald E. Knuth
Department of Computer Science
Stanford University
Stanford, CA 94305

Webless Literate Programming

Jim Fox

Abstract

This article introduces c-~~web~~ (*no-web*, for short) as an alternative to the CWEB 'literate programming' system. c-~~web~~ is a method which allows a programmer to both `tex` (format) and `cc` (compile) the same source, without the need for preprocessors.

What is c-~~web~~

In **c** all comments begin with the characters '/*' and end with the characters '*/'. c-~~web~~ is a macro package that TEXs all comments, 'verbatims' all the code, and uses the comment delimiters to switch between the two modes. A c-~~web~~ program can be compiled directly by **c** and can be formatted directly by TEX. It has the advantage of high portability, while providing fully TEX'd comments, page headers and footers, and a table of contents.

`\title{ ... }` Titles the program.

`\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.

`\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.

`*< ... >*` Prints verbatim. This allows **c** code to be included in comments.

Figure 1: c-~~web~~ definitions

Why c-~~web~~?

CWEB is essentially a **c** implementation of Edsger Dijkstra's *Notes on Structured Programming*, with fine formatting thrown in for good measure. The benefits of WEB are well known but it is unsuitable for many programmers and applications for a couple of reasons.

The first problem concerns portability. A program written in CWEB can only be conveniently implemented on a computer which already runs TEX. That is unfortunately a very small subset of the computing world. Anyone writing in CWEB greatly limits the portability of his or her programs.

The second problem concerns the translation of the code part of a program. A well written program consists of small pieces of code consisting of a documentation part, which explains to humans what the part does and how it does it, and a code part, which is a realization of the documentation. Both CWEB and c-~~web~~ print program listings assuming this method, and they both TEX the commentary. Where they differ is in the formatting of the code. c-~~web~~ leaves it alone except for indentation. CWEB gratuitously translates it into something that looks more like mathematics. Because programs undergo continual modifications, one tends to look to the source file to see what the code actually does. Many programmers, myself included, are more comfortable with code in the file that looks like the code in the listing.

```
/* sample.c in cnoweb format
   by Jim Fox, August 19, 1990
   \input cnoweb
   \title {Sample with procedure} */
        .
        .
/* \section{Sum}
    This procedure computes and returns
    $$ {\bf sum} = \sum_{i=0}^{\bf n}
       {\bf f}(i)$$

    There is no error checking
    in this example. */

double sum(f,n)
double (*f)();   /* function to call */
int n;           /* summation limit */
{
  int i;
  double s = 0;

   for (i=0; i<=n; i++) {
    s += f(i);
   }
  return (s);
}
     .
     .
/* \endc */
```

Figure 2: Procedure **sum** from program **sample**.

CWEB allows a programmer to break programs into small pieces without resorting to **c**'s procedure calls. This is an attempt to directly implement the 'layers' described by Dijkstra. c-~~web~~ cannot do this. However, procedures are often the better choice. They are more easily tested, more formally isolated from the caller, and usually produce more flexible code.

In any case, my effort here is only to introduce an alternate 'literate programming' method—not to compare the two beyond this introduction.

Using c-~~web~~

The c-~~web~~ program must begin with a comment that contains:

```
\input cnoweb
```

and must end with a comment that contains:

```
\endc
```

sum(f,n) **sample – 9**

/* **sum(f,n)** This procedure computes and returns

$$\mathbf{sum} = \sum_{i=0}^{\mathbf{n}} \mathbf{f}(i)$$

There is no error checking in this example. */

```
double sum(f,n)
double (*f)();      /* function to call */
int n;              /* summation limit */
{
    int i;
    double s = 0;

    for (i=0; i<=n; i++) {
        s += f(i);
    }
    return (s);
}
```

Figure 3: **sum** from the listing of **sample.c**

Other than this the program need not contain any TEX text. Most programs, however, will use plain TEX commands in comments, as well as several new commands provided by c-~~web~~. These are described in Figure 1.

Figure 2 is a sample procedure, **sum**, from a program in c-~~web~~ format. Figure 3 is the listing of the procedure. Not shown in figure 3 is the title page, which includes the title, synopsis (none in this example), and table of contents.

Features of c-~~web~~, some of which are demonstrated in the example, include:

1. Page breaks occur only before comments.
2. The code portion is printed not quite verbatim—indentation is automatically provided. Lines following an opening bracket or parenthesis are indented until the line containing the closing bracket or indentation.
3. The page heading contains the **c** file name, the page number, and the current section name.
4. Alignment rules on the top and left help verify indentation.

Trying it out

A sample program (`pf.c`) demonstrates c-~~web~~ and describes the commands in more detail. Interested persons should obtain a copy of the macro file (`cnoweb.tex`) and the sample program by anonymous ftp to `u.washington.edu`. They are in the directory `pub/tex/cnoweb`. Anyone without access to ftp may request the files by mail to me at the address below.

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

A TEX Previewer for "Slow" Terminals

Harold T. Stokes

In our department, we have a multi-user computer cluster with one copy of TEX and one laser printer. Our previewer sends document pages to the user's terminal in the Tektronix 4010/4014 graphics format. Most graphics terminals can emulate the Tektronix 4010/4014. This includes PCs using a terminal emulator like MSKermit.

The terminals in our department are connected to the system through a data switch. The rate at which data can be sent to a terminal is limited to 9600 baud (1200 bytes/second). This is rather slow for displaying graphics.

As an example, consider how we might draw the character T on the screen. The individual pixels for this character (from the `cmr10` font) are shown in Fig. 1*a*. Since Tektronix 4010/4014 graphics is vector-oriented, we might try the obvious raster scan shown in Fig. 1*b*. However, there are 44 line segments in that raster scan. To draw a single line segment in Tektronix 4010/4014 graphics, at least seven bytes must be sent to the terminal (sometimes eight or nine). At 9600 baud, it would require at least 0.25 seconds to display this single character. An entire document page which may contain 3000 characters would require more than ten minutes to be displayed. This, of course, is unacceptable for a previewer.

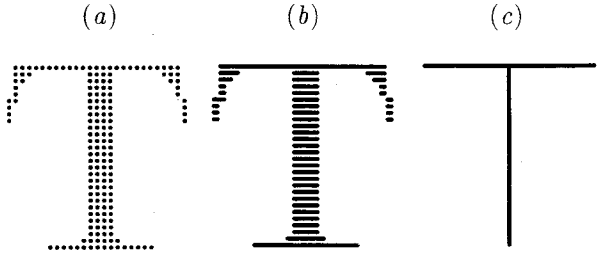

Fig. 1. The character T from `cmr10`: (*a*) pixels, (*b*) raster scan, (*c*) stick figure.