The CWEB System of Structured Documentation

(Version 3.0)

Donald E. Knuth and Silvio Levy

The CWEB System of Structured Documentation

Donald E. Knuth and Silvio Levy

This document describes a version of Don Knuth's WEB system, adapted to C by Silvio Levy. Since its creation in 1987, CWEB has been revised and enhanced in various ways, by both Knuth and Levy. We now believe that its evolution is near an end; however, bug reports, suggestions and comments are still welcome, and should be sent to Levy (levy@geom.umn.edu).

Readers who are familiar with Knuth's memo "The WEB System of Structured Documentation" will be able to skim this material rapidly, because CWEB and WEB share the same philosophy and (essentially) the same syntax. In some respects CWEB is a simplification of WEB: for example, CWEB does not need WEB's features for macro definition and string handling, because C and its preprocessor already take care of macros and strings. Similarly, the WEB conventions of denoting octal and hexadecimal constants by `@'77` and `@"3f` are replaced by C's conventions `077` and `0x3f`. All other features of WEB have been retained, and new features have been added.

We thank all who contributed suggestions and criticism to the development of CWEB. We are especially grateful to Steve Avery, Nelson Beebe, Hans-Hermann Bode, Klaus Guntermann, Norman Ramsey and Joachim Schnitter, who contributed code, and to Cameron Smith, who made many suggestions improving the manual. Ramsey has made literate programming accessible to users of yet other languages by means of his SPIDER system [see *Communications of the ACM* **32** (1989), 1051–1055]. Bode adapted CWEB so that it works for C++ as well, so in the text below you can read C++ for C if you so desire.

Introduction

The philosophy behind CWEB is that programmers who want to provide the best possible documentation for their programs need two things simultaneously: a language like TEX for formatting, and a language like C for programming. 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.

The structure of a software program may be thought of as a "web" that is made up of many interconnected pieces. To document such a program, we want to explain each individual part of the web and how it relates to its neighbors. The typographic tools provided by TEX give us an opportunity to explain the local structure of each part by making that structure visible, and the programming tools provided by C make it possible for us to specify the algorithms formally and unambiguously. By combining the two, we can develop a style of programming that maximizes our ability to perceive the structure of a complex piece of software, and at the same time the documented programs can be mechanically translated into a working software system that matches the documentation.

The CWEB system consists of two programs named CWEAVE and CTANGLE. When writing a CWEB program the user keeps the C code and the documentation in the same file, called the CWEB file and generally named `something.w`. The command '`cweave something`' creates an output file `something.tex`, which can then be fed to TEX, yielding a "pretty printed" version of `something.w` that correctly handles typographic details like page layout and the use of indentation, italics, boldface, and mathematical symbols. The typeset output also includes extensive cross-index information that is gathered automatically. Similarly, if you run the command '`ctangle something`' you will get a C file `something.c`, which can then be compiled to yield executable code.

Besides providing a documentation tool, CWEB enhances the C language by providing the ability to permute pieces of the program text, so that a large system can be understood entirely in terms of small sections and their local interrelationships. The CTANGLE program is so named because it takes a given web and moves the sections from their web structure into the order required by C; the advantage of programming in CWEB is that the algorithms can be expressed in "untangled" form, with each section explained separately. The CWEAVE program is so named because it takes a given web and intertwines the TEX and C portions contained in each section, 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 "*webe*", and the corresponding Latin imperative is "*texe*"!

A user of CWEB should be fairly familiar with the C programming language. A minimal amount of acquaintance with TEX is also desirable, but in fact it can be acquired as one uses CWEB, since straight text can be typeset in TeX with virtually no knowledge of that language. To someone familiar with both C and TeX the amount of effort necessary to learn the commands of `CWEB` is small.

Overview

Two kinds of material go into `CWEB` files: TeX text and C text. A programmer writing in `CWEB` should be thinking both of the documentation and of the C program being created; i.e., the programmer should be instinctively aware of the different actions that `CWEAVE` and `CTANGLE` will perform on the `CWEB` file. TeX text is essentially copied without change by `CWEAVE`, and it is entirely deleted by `CTANGLE`; the TeX text is "pure documentation." C text, on the other hand, is formatted by `CWEAVE` and it is shuffled around by `CTANGLE`, according to rules that will become clear later. For now the important point to keep in mind is that there are two kinds of text. Writing `CWEB` programs is something like writing TeX documents, but with an additional "C mode" that is added to TeX's horizontal mode, vertical mode, and math mode.

A `CWEB` file is built up from units called *sections* that are more or less self-contained. Each section has three parts:

- A TeX part, containing explanatory material about what is going on in the section.
- A middle part, containing macro definitions that serve as abbreviations for C constructions that would be less comprehensible if written out in full each time. They are turned by `CTANGLE` into preprocessor macro definitions.
- A C part, containing a piece of the program that `CTANGLE` will produce. This C code should ideally be about a dozen lines long, so that it is easily comprehensible as a unit and so that its structure is readily perceived.

The three parts of each section must appear in this order; i.e., the TeX commentary must come first, then the middle part, and finally the C code. Any of the parts may be empty.

A section begins with either of the symbols '`@␣`' or '`@*`', where '␣' denotes a blank space. A section ends at the beginning of the next section (i.e., at the next '`@␣`' or '`@*`'), or at the end of the file, whichever comes first. The `CWEB` file may also contain material that is not part of any section at all, namely the text (if any) that occurs before the first section. Such text is said to be "in limbo"; it is ignored by `CTANGLE` and copied essentially verbatim by `CWEAVE`, so its function is to provide any additional formatting instructions that may be desired in the TeX output. Indeed, it is customary to begin a `CWEB` file with TeX code in limbo that loads special fonts, defines special macros, changes the page sizes, and/or produces a title page.

Sections are numbered consecutively, starting with 1. These numbers appear at the beginning of each section of the TeX documentation output by `CWEAVE`, and they appear as bracketed comments at the beginning and end of the code generated by that section in the C program output by `CTANGLE`.

Section Names

Fortunately, you never mention these numbers yourself when you are writing in `CWEB`. You just say '`@␣`' or '`@*`' at the beginning of each new section, and the numbers are supplied automatically by `CWEAVE` and `CTANGLE`. As far as you are concerned, a section has a *name* instead of a number; its name is specified by writing '`@<`' followed by TeX text followed by '`@>`'. When `CWEAVE` outputs a section name, it replaces the '`@<`' and '`@>`' by angle brackets and inserts the section number in small type. Thus, when you read the output of `CWEAVE` it is easy to locate any section that is referred to in another section.

For expository purposes, a section name should be a good description of the contents of that section; i.e., it should stand for the abstraction represented by the section. Then the section can be "plugged into" one or more other sections in such a way that unimportant details of its inner workings are suppressed. A section name therefore ought to be long enough to convey the necessary meaning.

Unfortunately, it is laborious to type such long names over and over again, and it is also difficult to specify a long name twice in exactly the same way so that `CWEAVE` and `CTANGLE` will be able to match the names to the sections. To ameliorate this situation, `CWEAVE` and `CTANGLE` let you abbreviate a section name, so long as the full name appears somewhere in the `CWEB` file; you can type simply '`@<`α`...@>`', where α is any string that is a prefix of exactly one section name appearing in the file. For example, '`@<Clear the arrays@>`' can be abbreviated to '`@<Clear...@>`' if no other section name begins with the five letters '`Clear`'. Elsewhere you might use the abbreviation '`@<Clear t...@>`', and so on.

Section names must otherwise match character for character, except that consecutive characters of white space (spaces, tab marks, newlines, and/or form feeds) are treated as equivalent to a single space, and such spaces are deleted at the beginning and end of the name. Thus, ‘`@< Clear  the arrays @>`’ will also match the name in the previous example. Spaces following the ellipsis in abbreviations are ignored as well, but not those before, so that ‘`@<Clear t ...@>`’ would not match ‘`@<Clear the arrays@>`’.

What CTANGLE Does

We have said that a section begins with ‘`@␣`’ or ‘`@*`’, but we didn't say how it gets divided up into a TeX part, a middle part, and a C part. The middle part begins with the first appearance of ‘`@d`’ or ‘`@f`’ in the section, and the C part begins with the first appearance of ‘`@c`’ or ‘`@<section name@>=`’. In the latter case you are saying, in effect, that the section name stands for the C text that follows. Alternatively, if the C part begins with ‘`@c`’ instead of a section name, the current section is said to be *unnamed*.

The construct ‘`@<section name@>`’ can appear any number of times in the C part of a section: subsequent appearances indicate that the named section is being “used” rather than “defined”, that is, that the C code for the named section, presumably defined elsewhere, should be spliced in at this point in the C program. Indeed, the main idea of CTANGLE is to make a C program out of individual sections, named and unnamed. The exact way in which this is done is this: First all the macro definitions indicated by ‘`@d`’ are turned into C preprocessor macro definitions and copied at the beginning. Then the C parts of unnamed sections are copied down, in order; this constitutes the first-order approximation to the text of the program. (There should be at least one unnamed section, otherwise there will be no program.) Then all section names that appear in the first-order approximation are replaced by the C parts of the corresponding sections, and this substitution process continues until no section names remain. All comments are removed, because the C program is intended only for the eyes of the C compiler.

If the same name has been given to more than one section, the C text for that name is obtained by putting together all of the C parts in the corresponding sections. This feature is useful, for example, in a section named ‘Global variables’, since one can then declare global variables in whatever sections those variables are introduced. When several sections have the same name, CWEAVE assigns the first section number as the number corresponding to that name, and it inserts a note at the bottom of that section telling the reader to ‘See also sections so-and-so’; this footnote gives the numbers of all the other sections having the same name as the present one. The C text corresponding to a section is usually formatted by CWEAVE so that the output has an equivalence sign in place of the equals sign in the CWEB file; i.e., the output says ‘$\langle$ section name $\rangle \equiv$ C text’. However, in the case of the second and subsequent appearances of a section with the same name, this ‘$\equiv$’ sign is replaced by ‘$+\equiv$’, as an indication that the following C text is being appended to the C text of another section.

As CTANGLE enters and leaves sections, it inserts preprocessor `#line` commands into the C output file. This means that when the compiler gives you error messages, or when you debug your program, the messages refer to line numbers in the CWEB file, and not in the C file. In most cases you can therefore forget about the C file altogether.

What CWEAVE Does

The general idea of CWEAVE is to make a `.tex` file from the CWEB file in the following way: The first line of the `.tex` file tells TeX to input a file with macros that define CWEB's documentation conventions. The next lines of the file will be copied from whatever TeX text is in limbo before the first section. Then comes the output for each section in turn, possibly interspersed with end-of-page marks. Finally, CWEAVE will generate a cross-reference index that lists each section number in which each C identifier appears, and it will also generate an alphabetized list of the section names, as well as a table of contents that shows the page and section numbers for each “starred” section.

What is a “starred” section, you ask? A section that begins with ‘`@*`’ instead of ‘`@␣`’ is slightly special in that it denotes a new major group of sections. The ‘`@*`’ should be followed by the title of this group, followed by a period. Such sections will always start on a new page in the TeX output, and the group title will appear as a running headline on all subsequent pages until the next starred section. The title will also appear in the table of contents, and in boldface type at the beginning of its section. Caution: Do not use TeX control sequences in such titles, unless you know that the `cwebmac` macros will do the right thing with them. The reason is that these titles are converted to uppercase when they appear as running heads, and they are converted to boldface when they appear at the beginning of their sections, and they are also written out to a table-of-contents file used for temporary storage while TeX is working; whatever control sequences you use must be meaningful in all three of these modes.

The TeX output produced by CWEAVE for each section consists of the following: First comes the section number (e.g., '\M123.' at the beginning of section 123, except that '\N' appears in place of '\M' at the beginning of a starred section). Then comes the TeX part of the section, copied almost verbatim except as noted below. Then comes the middle part and the C part, formatted so that there will be a little extra space between them if both are nonempty. The middle and C parts are obtained by inserting a bunch of funny-looking TeX macros into the C program; these macros handle typographic details about fonts and proper math spacing, as well as line breaks and indentation.

C Code in TeX Text and Vice Versa

When you are typing TeX text, you will probably want to make frequent reference to variables and other quantities in your C code, and you will want those variables to have the same typographic treatment when they appear in your text as when they appear in your program. Therefore the CWEB language allows you to get the effect of C editing within TeX text, if you place '|' marks before and after the C material. For example, suppose you want to say something like this:

If *pa* is declared as '**int** $*pa$', the assignment $pa = \&a[0]$ makes *pa* point to the zeroth element of a.

The TeX text would look like this in your CWEB file:

```
If |pa| is declared as `|int *pa|', the
assignment |pa=&a[0]| makes |pa| point to the zeroth element of |a|.
```

And CWEAVE translates this into something you are glad you didn't have to type:

```
If \\{pa} is declared as `\&{int} ${}{*}\\{pa}$',
the assignment $\\{pa}\K{\AND}\|a[\T{0}]$
makes \\{pa} point to the zeroth element of \|a.
```

Incidentally, the cross-reference index that CWEAVE would make, in the presence of a comment like this, would include the current section number as one of the index entries for *pa*, even though *pa* might not appear in the C part of this section. Thus, the index covers references to identifiers in the explanatory comments as well as in the program itself; you will soon learn to appreciate this feature. However, the identifiers **int** and a would not be indexed, because CWEAVE does not make index entries for reserved words or single-letter identifiers. Such identifiers are felt to be so ubiquitous that it would be pointless to mention every place where they occur.

Although a section begins with TeX text and ends with C text, we have noted that the dividing line isn't sharp, since C text can be included in TeX text if it is enclosed in '|...|'. Conversely, TeX text appears frequently within C text, because everything in comments (i.e., between /* and */, or following //) is treated as TeX text. Likewise, the text of a section name consists of TeX text, but the construct @<section name@> as a whole is is expected to be found in C text; thus, one typically goes back and forth between the C and TeX environments in a natural way, as in these examples:

```
if (x==0) @<Empty the |buffer| array@>
... using the algorithm in |@<Empty the |buffer| array@>|.
```

The first of these excerpts would be found in the C part of a section, into which the code from the section named "Empty the *buffer* array" is being spliced. The second excerpt would be found in the TeX part of the section, and the named section is being "cited", rather than defined or used. (Note the '|...|' surrounding the section name in this case.)

Macros

The control code @d followed by

identifier C text   or by   *identifier* $(par_1, \ldots, par_n)$ C text

(where there is no blank between the *identifier* and the parentheses in the second case) is transformed by CTANGLE into a preprocessor command, starting with #define, which is printed at the top of the C output file as explained earlier.

A '@d' macro definition can go on for several lines, and the newlines don't have to be protected by backslashes, since CTANGLE itself inserts the backslashes. If for any reason you need a #define command at a specific spot in your C file, you can treat it as C code, instead of as a CWEB macro; but then you do have to protect newlines yourself.

Strings and constants

If you want a string to appear in the C file, delimited by pairs of ' or " marks as usual, you can type it exactly so in the CWEB file, except that the character '@' should be typed '@@' (it becomes a control code, the only one that can appear in strings; see below). Strings should end on the same line as they begin, unless there's a backslash at the end of lines within them.

TeX and C have different ways to refer to octal and hex constants, because TeX is oriented to technical writing while C is oriented to computer processing. In TeX you make a constant octal or hexadecimal by prepending ' or ", respectively, to it; in C the constant should be preceded by 0 or 0x. In CWEB it seems reasonable to let each convention hold in its respective realm; so in C text you get 40_8 by typing '040', which CTANGLE faithfully copies into the C file (for the compiler's benefit) and which CWEAVE prints as $^\circ 40$. Similarly, CWEAVE prints the hexadecimal C constant '0x20' as #20. The use of italic font for octal digits and typewriter font for hexadecimal digits makes the meaning of such constants clearer in a document. For consistency, then, you should type '|040|' or '|0x20|' in the TeX part of the section.

Control codes

A CWEB *control code* is a two-character combination of which the first is '@'. We've already seen the meaning of several control codes; it's time to list them more methodically.

In the following list, the letters in brackets after a control code indicate in what contexts that code is allowed. L indicates that the code is allowed in limbo; T (for TeX), M (for middle) and C mean that the code is allowed in each of the three parts of a section, at top level—that is, outside such constructs as '|...|' and section names. An arrow $\to$ means that the control code terminates the present part of the CWEB file, and inaugurates the part indicated by the letter following the arrow. Thus $[LTMC \to T]$ next to @␣ indicates that this control code can occur in limbo, or in any of the three parts of a section, and that it starts the (possibly empty) TeX part of the following section.

Two other abbreviations can occur in these brackets: The letter r stands for *restricted context*, that is, material inside C comments, section names, C strings and control texts (defined below); the letter c stands for *inner* C *context*, that is, C material inside '|...|' (including '|...|'s inside comments, but not those occurring in other restricted contexts). An asterisk $*$ following the brackets means that the context from this control code to the matching @> is restricted.

Control codes involving letters are case-insensitive: thus @d and @D are equivalent. Only the lowercase versions are mentioned specifically below.

@@ $[LTMCrc]$ A double @ denotes the single character '@'. This is the only control code that is legal everywhere. Note that you must use this convention if you are giving an internet email address in a CWEB file (e.g., levy@@geom.umn.edu).

Here are the codes that introduce the TeX part of a section.

@␣ $[LTMC \to T]$ This denotes the beginning of a new (unstarred) section. A tab mark or form feed or end-of-line character is equivalent to a space when it follows an @ sign (and in most other cases).

@* $[LTMC \to T]$ This denotes the beginning of a new starred section, i.e., a section that begins a new major group. The title of the new group should appear after the @*, followed by a period. As explained above, TeX control sequences should be avoided in such titles unless they are quite simple. When CWEAVE and CTANGLE read a @*, they print an asterisk on the terminal followed by the current section number, so that the user can see some indication of progress. The very first section should be starred.

You can specify the "depth" of a starred section by typing * or a decimal number after the @*; this indicates the relative ranking of the current group of sections in the program hierarchy. Top-level portions of the program, introduced by @**, get their names typeset in boldface type in the table of contents; they are said to have depth -1. Otherwise the depth is a nonnegative number, which governs the amount of indentation on the contents page. Such indentation helps clarify the structure of a long program. The depth is assumed to be 0 if it is not specified explicitly; when your program is short, you might as well leave all depths zero. A starred section always begins a new page in the output, unless the depth is greater than 1.

The middle part of each section consists of any number of macro definitions (beginning with `@d`) and format definitions (beginning with `@f` or `@s`), intermixed in any order.

`@d` $[TM \to M]$ Macro definitions begin with `@d`, followed by an identifier and optional parameters and C text as explained earlier.

`@f` $[TM \to M]$ Format definitions begin with `@f`; they cause `CWEAVE` to treat identifiers in a special way when they appear in C text. The general form of a format definition is '`@f` l r', followed by an optional comment enclosed between `/*` and `*/`, where l and r are identifiers; `CWEAVE` will subsequently treat identifier l as it currently treats r. This feature allows a `CWEB` programmer to invent new reserved words and/or to unreserve some of C's reserved identifiers. If r is the special identifier '*TeX*', identifier l will be formatted as a TeX control sequence; for example, '`@f foo TeX`' in the `CWEB` file will cause identifier *foo* to be output as `\foo` by `CWEAVE`. The programmer should define `\foo` to have whatever custom format is desired, assuming TeX math mode. (Each underline character is converted to `x` when making the TeX control sequence; thus *foo_bar* becomes `\fooxbar`. Other characters, including digits, are left untranslated, so TeX will consider them as macro parameters, not as part of the control sequence itself. For example,

```
\def\x#1{x_{#1}} @f x1 TeX @f x2 TeX
```

will format `x1` and `x2` not as *x1* and *x2* but as x_1 and x_2.

`CWEAVE` knows that identifiers being defined with a **typedef** should become reserved words; thus you don't need format definitions very often.

`@s` $[TM \to M;\ L]$ Same as `@f`, but `CWEAVE` does not show the format definition in the output, and the optional C comment is not allowed. This is used mostly in `@i` files.

Next come the codes that govern the C part of a section.

`@c @p` $[TM \to C]$ The C part of an unnamed section begins with `@c` (or with `@p` for "program"; both control codes do the same thing). This causes `CTANGLE` to append the following C code to the first-order program text, as explained on page 3. Note that `CWEAVE` does not print a '`@c`' in the TeX output, so if you are creating a `CWEB` file based on a TeX-printed `CWEB` documentation you have to remember to insert `@c` in the appropriate places of the unnamed sections.

`@<` $[TM \to C;\ C;\ c]$ $*$ This control code introduces a section name (or unambiguous prefix, as discussed above), which consists of TeX text and extends to the matching `@>`. The whole construct `@<...@>` is conceptually a C element. The behavior is different depending on the context:

A `@<` appearing in contexts T and M attaches the following section name to the current section, and inaugurates the C part of the section. The closing `@>` should be followed by `=` or `+=`.

In context C, `@<` indicates that the named section is being used—its C definition is spliced in by `CTANGLE`, as explained on page 3. As an error-detection measure, `CTANGLE` and `CWEAVE` complain if such a section name is followed by `=`, because most likely this is meant as the definition of a new section, and so should be preceded by `@␣`. If you really want to say $\langle\,\text{foo}\,\rangle = bar$, where $\langle\,\text{foo}\,\rangle$ is being used and not defined, put a newline before the `=`.

Finally, in inner C context (that is, within '`|...|`' in the TeX part of a section or in a comment), `@<...@>` means that the named section is being cited. Such an occurrence is ignored by `CTANGLE`. Note that even here we think of the section name as being a C element, hence the `|...|`.

`@(` $[TM \to C;\ C;\ c]$ $*$ A section name can begin with `@(`. Everything works just as for `@<`, except that the C code of the section named `@(foo@>` is written by `CTANGLE` to file `foo`. In this way you can get multiple-file output from a single `CWEB` file. (The `@d` definitions are not output to such files, only to the master `.c` file.) One use of this feature is to produce header files for other program modules that will be loaded with the present one. Another use is to produce a test routine that goes with your program. By keeping the sources for a program and its header and test routine together, you are more likely to keep all three consistent with each other. Notice that the output of a named section can be incorporated in several different output files, because you can mention `@<foo@>` in both `@(bar1@>` and `@(bar2@>`.

`@h` $[Cc]$ Causes `CTANGLE` to insert at the current spot the `#define` statements from the middle parts of all sections, and *not* to write them at the beginning of the C file. Useful when you want the macro definitions to come after the include files, say. (Ignored by `CTANGLE` inside '`|...|`'.)

The next several control codes introduce "control texts", which end with the next '`@>`'. The closing '`@>`' must be on the same line of the CWEB file as the line where the control text began. The context from each of these control codes to the matching `@>` is restricted.

`@^` [*TMCc*] ∗ The control text that follows, up to the next '`@>`', will be entered into the index together with the identifiers of the C program; this text will appear in roman type. For example, to put the phrase "system dependencies" into the index that is output by CWEAVE, type '`@^system dependencies@>`' in each section that you want to index as system dependent.

`@.` [*TMCc*] ∗ The control text that follows will be entered into the index in `typewriter type`.

`@:` [*TMCc*] ∗ The control text that follows will be entered into the index in a format controlled by the TEX macro '`\9`', which you should define as desired.

`@t` [*MCc*] ∗ The control text that follows will be put into a TEX `\hbox` and formatted along with the neighboring C program. This text is ignored by CTANGLE, but it can be used for various purposes within CWEAVE. For example, you can make comments that mix C and classical mathematics, as in '$size < 2^{15}$', by typing '`|size < 2@t$^{15}$@>|`'.

`@=` [*MCc*] ∗ The control text that follows will be passed verbatim to the C program.

`@q` [*LTMCc*] ∗ The control text that follows will be totally ignored—it's a comment for readers of the CWEB file only. A file intended to be included in limbo, with `@i`, can identify itself with `@q` comments. Another use is to balance unbalanced parentheses in C strings, so that your text editor's parenthesis matcher doesn't go into a tailspin.

`@!` [*TMCc*] ∗ The section number in an index entry will be underlined if '`@!`' immediately precedes the identifier or control text being indexed. This convention is used to distinguish the sections where an identifier is defined, or where it is explained in some special way, from the sections where it is used. A reserved word or an identifier of length one will not be indexed except for underlined entries. An '`@!`' is implicitly inserted by CWEAVE when an identifier is being defined or declared in C code; for example, the definition

$$\textbf{int } \mathit{array}[\mathit{max\_dim}],\ \mathit{count} = \mathit{old\_count};$$

makes the names *array* and *count* get an underlined entry in the index. Statement labels, function definitions like *main*(**int** *argc*, **char** ∗*argv*[]), and **typedef** definitions also imply underlining. An old-style function definition (without prototyping) doesn't define its arguments; the arguments will, however, be considered to be defined (i.e., their index entries will be underlined) if their types are declared before the body of the function in the usual way (e.g., '**int** *argc*; **char** ∗*argv*[]; { . . . }').

We now turn to control codes that affect only the operation of CTANGLE.

`@'` [*MCc*] This control code is dangerous because it has quite different meanings in CWEB and the original WEB. In CWEB it produces the decimal constant corresponding to the ASCII code for a string of length 1 (e.g., `@'a'` is CTANGLEd into `97` and `@'\t'` into `9`). You might want to use this if you need to work in ASCII on a non-ASCII machine; but in most cases the C conventions of `<ctype.h>` are adequate for character-set-independent programming.

`@&` [*MCc*] The `@&` operation causes whatever is on its left to be adjacent to whatever is on its right, in the C output. No spaces or line breaks will separate these two items.

`@l` [*L*] CWEB programmers have the option of using any 8-bit character code from the often-forbidden range 128–255 within TEX text; such characters are also permitted in strings and even in identifiers of the C program. Under various extensions of the basic ASCII standard, the higher 8-bit codes correspond to accented letters, letters from non-Latin alphabets, and so on. When such characters occur in identifiers, CTANGLE must replace them by standard ASCII alphanumeric characters or `_`, in order to generate legal C code. It does this by means of a transliteration table, which by default associates the string `Xab` to the character with ASCII code $^{\#}ab$ (where a and b are hexadecimal digits, and $a \geq 8$). By placing the construction `@l␣ab␣newstring` in limbo, you are telling CTANGLE to replace this character by `newstring` instead. For example, the ISO Latin-1 code for the letter 'ü' is `#FC` (or `'\374'`), and CTANGLE will normally change this code to the three-character sequence `XFC` if it appears in an identifier. If you say `@l fc ue`, the code will be transliterated into `ue` instead.

CWEAVE passes 8-bit characters straight through to TEX without transliteration; therefore TEX must be prepared to receive them. If you are formatting all your nonstandard identifiers as "custom" control sequences, you should make TEX treat all their characters as letters. Otherwise you should either make your 8-bit codes "active" in TEX, or load fonts that contain the special characters you need in the correct positions. (The font selected by TEX control sequence `\it` is used for identifiers.) Look for special macro packages designed for CWEB users in your language; or, if you are brave, write one yourself.

The next eight control codes (namely '`@,`', '`@/`', '`@|`', '`@#`', '`@+`', '`@;`', '`@[`', and '`@]`') have no effect on the C program output by CTANGLE; they merely help to improve the readability of the TEX-formatted C that is output by CWEAVE, in unusual circumstances. CWEAVE's built-in formatting method is fairly good when dealing with syntactically correct C text, but it is incapable of handling all possible cases, because it must deal with fragments of text involving macros and section names; these fragments do not necessarily obey C's syntax. Although CWEB allows you to override the automatic formatting, your best strategy is not to worry about such things until you have seen what CWEAVE produces automatically, since you will probably need to make only a few corrections when you are touching up your documentation.

`@,` [*MCc*] This control code inserts a thin space in CWEAVE's output. Sometimes you need this extra space if you are using macros in an unusual way, e.g., if two identifiers are adjacent.

`@/` [*MC*] This control code causes a line break to occur within a C program formatted by CWEAVE. Line breaks are chosen automatically by TEX according to a scheme that works 99% of the time, but sometimes you will prefer to force a line break so that the program is segmented according to logical rather than visual criteria. If a comment follows, say '`@/@,`' to break the line before the comment.

`@|` [*MC*] This control code specifies an optional line break in the midst of an expression. For example, if you have a long expression on the right-hand side of an assignment statement, you can use '`@|`' to specify breakpoints more logical than the ones that TEX might choose on visual grounds.

`@#` [*MC*] This control code forces a line break, like `@/` does, and it also causes a little extra white space to appear between the lines at this break. You might use it, for example, between groups of macro definitions that are logically separate but within the same section. CWEB automatically inserts this extra space between functions, between external declarations and functions, and between declarations and statements within a function.

`@+` [*MC*] This control code cancels a line break that might otherwise be inserted by CWEAVE, e.g., before the word '**else**', if you want to put a short if–else construction on a single line. If you say '`{@+`' at the beginning of a compound statement that is the body of a function, the first declaration or statement of the function will appear on the same line as the left brace, and it will be indented by the same amount as the second declaration or statement on the next line.

`@;` [*MC*] This control code is treated like a semicolon, for formatting purposes, except that it is invisible. You can use it, for example, after a section name or macro when the C text represented by that section or macro is a compound statement or ends with a semicolon. Consider constructions like

```
if (condition) macro @;
else break;
```

where *macro* is defined to be a compound statement (enclosed in braces). This is a well-known infelicity of C syntax.

`@[` [*MC*] See `@]`.

`@]` [*MC*] Place `@[...@]` brackets around program text that CWEAVE is supposed to format as an expression, if it doesn't already do so. (This occasionally applies to unusual macro arguments.) Also insert '`@[@]`' between a simple type name and a left parenthesis when declaring a pointer to a function, as in

```
int @[@] (*f)();
```

otherwise CWEAVE will confuse the first part of that declaration with the C++ expression '**int**$(*f)$'.

The remaining control codes govern the input that CWEB sees.

`@x @y @z` [*change_file*] CWEAVE and CTANGLE are designed to work with two input files, called *web_file* and *change_file*, where *change_file* contains data that overrides selected portions of *web_file*. The resulting merged text is actually what has been called the CWEB file elsewhere in this report.

Here's how it works: The change file consists of zero or more "changes," where a change has the form '`@x`⟨old lines⟩`@y`⟨new lines⟩`@z`'. The special control codes `@x`, `@y`, `@z`, which are allowed only in change files, must appear at the beginning of a line; the remainder of such a line is ignored. The ⟨old lines⟩ represent material that exactly matches consecutive lines of the *web_file*; the ⟨new lines⟩ represent zero or more lines that are supposed to replace the old. Whenever the first "old line" of a change is found to match a line in the *web_file*, all the other lines in that change must match too.

Between changes, before the first change, and after the last change, the change file can have any number of lines that do not begin with '`@x`', '`@y`', or '`@z`'. Such lines are bypassed and not used for matching purposes.

This dual-input feature is useful when working with a master CWEB file that has been received from elsewhere (e.g., `tangle.w` or `weave.w` or `tex.web`), when changes are desirable to customize the program for your local computer system. You will be able to debug your system-dependent changes without clobbering the master web file; and once your changes are working, you will be able to incorporate them readily into new releases of the master web file that you might receive from time to time.

`@i` [*web_file*] Furthermore the *web_file* itself can be a combination of several files. When either CWEAVE or CTANGLE is reading a file and encounters the control code `@i` at the beginning of a line, it interrupts normal reading and starts looking at the file named after the `@i`, much as the C preprocessor does when it encounters an `#include` line. After the included file has been entirely read, the program goes back to the next line of the original file. The file name following `@i` can be surrounded by `"` characters, but such delimiters are optional. Include files can nest.

Change files can have lines starting with `@i`. In this way you can replace one included file with another. Conceptually, the replacement mechanism described above does its work first, and its output is then checked for `@i` lines. If `@i foo` occurs between `@y` and `@z` in a change file, individual lines of file `foo` and files it includes are not changeable; but changes can be made to lines from files that were included by unchanged input.

On UNIX systems (and others that support environment variables), if the environment variable CWEBINPUTS is set, or if the compiler flag of the same name was defined at compile time, CWEB will look for include files in the directory thus named, if it cannot find them in the current directory.

Additional features and caveats

1. In certain installations of CWEB that have an extended character set, the characters '↑', '↓', '→', '≠', '≤', '≥', '≡', '∨', '∧', '⊂', and '⊃' can be typed as abbreviations for '`++`', '`--`', '`->`', '`!=`', '`<=`', '`>=`', '`==`', '`||`', '`&&`', '`<<`', and '`>>`', respectively.

2. If you have an extended character set, you can use it with only minimal restrictions, as discussed under the rules for `@l` above. But you should stick to standard ASCII characters if you want to write programs that will be useful to all the poor souls out there who don't have extended character sets.

3. The TeX file output by CWEAVE is broken into lines having at most 80 characters each. When TeX text is being copied, the existing line breaks are copied as well. If you aren't doing anything too tricky, CWEAVE will recognize when a TeX comment is being split across two or more lines, and it will append '`%`' to the beginning of such continued comments.

4. C text is translated by a "bottom up" procedure that identifies each token as a "part of speech" and combines parts of speech into larger and larger phrases as much as possible according to a special grammar that is explained in the documentation of CWEAVE. It is easy to learn the translation scheme for simple constructions like single identifiers and short expressions, just by looking at a few examples of what CWEAVE does, but the general mechanism is somewhat complex because it must handle much more than C itself. Furthermore the output contains embedded codes that cause TeX to indent and break lines as necessary, depending on the fonts used and the desired page width. For best results it is wise to avoid enclosing long C texts in `|...|`, since the indentation and line breaking codes are omitted when the `|...|` text is translated from C to TeX. Stick to simple expressions or statements. If a C preprocessor command is enclosed in `|...|`, the `#` that introduces it must be at the beginning of a line, or CWEAVE won't print it correctly.

5. Comments are not permitted in `|...|` text. After a '`|`' signals the change from TeX text to C text, the next '`|`' that is not part of a string or control text or section name ends the C text.

6. A comment must have properly nested occurrences of left and right braces, otherwise CWEAVE will complain. But it does try to balance the braces, so that TeX won't foul up too much.

7. When you're debugging a program and decide to omit some of your C code, do NOT simply "comment it out." Such comments are not in the spirit of CWEB documentation; they will appear to readers as if they were explanations of the uncommented-out instructions. Furthermore, comments of a program must be valid

$\TeX$ text; hence `CWEAVE` will get confused if you enclose C statements in `/*...*/` instead of in `/*|...|*/`. If you must comment out C code, you can surround it with preprocessor commands like `#if 0==1` and `#endif`.

8. The `@f` feature allows you to define one identifier to act like another, and these format definitions are carried out sequentially. In general, a given identifier has only one printed format throughout the entire document, and this format is used even before the `@f` that defines it. The reason is that `CWEAVE` operates in two passes; it processes `@f`'s and cross-references on the first pass and it does the output on the second. (However, identifiers that implicitly get a boldface format, thanks to a `typedef` declaration, don't obey this rule: they are printed differently before and after the relevant `typedef`. This is unfortunate, but hard to fix. You can get around the restriction by using `@s`, before or after the `typedef`.)

9. Sometimes it is desirable to insert spacing into formatted C code that is more general than the thin space provided by '`@,`'. The `@t` feature can be used for this purpose; e.g., '`@t\hskip 1in@>`' will leave one inch of blank space. Furthermore, '`@t\4@>`' can be used to backspace by one unit of indentation, since the control sequence `\4` is defined in `cwebmac` to be such a backspace. (This control sequence is used, for example, at the beginning of lines that contain labeled statements, so that the label will stick out a little at the left.) You can also use '`@t}\3{-5@>`' to force a break in the middle of an expression.

10. Each identifier in `CWEB` has a single formatting convention. Therefore you shouldn't use the same identifier to denote, say, both a type name and part of a `struct`, even though C does allow this.

Running the programs

The UNIX command line for `CTANGLE` is

```
ctangle [options] web_file[.w] [{change_file[.ch]|-} [out_file]]
```

and the same conventions apply to `CWEAVE`. If '`-`' or no change file is specified, the change file is null. The extensions `.w` and `.ch` are appended only if the given file names contain no dot. If the web file defined in this way cannot be found, the extension `.web` will be tried. For example, '`cweave cob`' will try to read `cob.w`; failing that, it will try `cob.web` before giving up. If no output file name is specified, the name of the C file output by `CTANGLE` is obtained by appending the extension `.c`; the name of the $\TeX$ file output by `CWEAVE` gets the extension `.tex`. Index files output by `CWEAVE` replace `.tex` by `.idx` and `.scn`.

Programmers who like terseness might choose to set up their operating shell so that '`wv`' expands to '`cweave -bhp`'; this will suppress most terminal output from `CWEAVE` except for error messages.

Options are introduced either by a `-` sign, to turn an option off, or by a `+` sign to turn one on. For example, '`-fb`' turns off options `f` and `b`; '`+s`' turns on option `s`. Options can be specified before the file names, after the file names, or both. The following options are currently implemented:

- `b` Print a banner line at the beginning of execution. (On by default.)
- `f` Force line breaks after each C statement formatted by `CWEAVE`. (On by default; `-f` saves paper but looks less C-like to some people.) (Has no effect on `CTANGLE`.)
- `h` Print a happy message at the conclusion of a successful run. (On by default.)
- `p` Give progress reports as the program runs. (On by default.)
- `s` Show statistics about memory usage after the program runs to completion. (Off by default.) If you have large `CWEB` files or sections, you may need to see how close you come to exceeding the capacity of `CTANGLE` and/or `CWEAVE`.
- `x` Include indexes and a table of contents in the $\TeX$ file output by `CWEAVE`. (On by default.) (Has no effect on `CTANGLE`.)

Further details about formatting

You may not like the way `CWEAVE` handles certain situations. If you're desperate, you can customize `CWEAVE` by changing its grammar. This means changing the source code, a task that you might find amusing. A table of grammar rules appears in the `CWEAVE` source listing, and you can make a separate copy of that table by copying the file `prod.w` found in the `CWEB` sources and saying '`cweave -x prod`', followed by '`tex prod`'.

You can see exactly how `CWEAVE` is parsing your C code by preceding it with the line '`@ @c @2`'. (The control code '`@2`' turns on a "peeping" mode, and '`@0`' turns it off.) For example, if you run `CWEAVE` on the file

```
@ @c @2
main (argc,argv)
```

```
char **argv;
{ for (;argc>0;argc--) printf("%s\n",argv[argc-1]); }
```

you get the following gibberish on your screen:

```
[...]
4:*exp ( +exp+ )...
11:*exp +exp+ int...
5:*+exp+ int +unorbinop+...
[...]
60: +fn_decl+*+{+ -stmt- +}-
55:*+fn_decl+ -stmt-
52:*+function-
[...]
```

The first line says that grammar rule 4 has just been applied, and `CWEAVE` currently has in its memory a sequence of chunks of TEX code (called "scraps") that are respectively of type *exp* (for expression), open-parenthesis, *exp* again, close-parenthesis, and further scraps that haven't yet been considered by the parser. (The `+` and `-` signs stipulate that TEX should be in or out of math mode at the scrap boundaries. The `*` shows the parser's current position.) Then rule 11 is applied, and the sequence (*exp*) becomes an *exp* and so on. In the end the whole C text has become one big scrap of type *function*.

Sometimes things don't work as smoothly, and you get a bunch of lines lumped together. This means that `CWEAVE` could not digest something in your C code. For instance, suppose '`@<Argument definitions@>`' had appeared instead of '`char **argv;`' in the program above. Then `CWEAVE` would have been somewhat mystified, since it thinks that section names are just *exp*s. Thus it would tell TEX to format '⟨ Argument declarations 2 ⟩' on the same line as '*main*(*argc*, *argv*)'. In this case you should help `CWEAVE` by putting '`@/`' after '`main(argc,argv)`'.

`CWEAVE` automatically inserts a bit of extra space between declarations and the first apparent statement of a block. One way to defeat this spacing locally is

```
int x;@+@t}\6{@>
@<Other locals@>@;@#
```

the '`@#`' will put extra space after '⟨ Other locals ⟩'.

Appendices

As an example of a real program written in `CWEB`, Appendix A contains an excerpt from the `CWEB` program itself. The reader who examines the listings in this appendix carefully will get a good feeling for the basic ideas of `CWEB`.

Appendix B is the file that sets TEX up to accept the output of `CWEAVE`, and Appendix C discusses how to use some of those macros to vary the output formats.

A "long" version of this manual, which can be produced from the `CWEB` sources via the UNIX command `make fullmanual`, also contains appendices D, E, and F, which exhibit the complete source code for `CTANGLE` and `CWEAVE`.

Appendix A: Excerpts from a CWEB Program

This appendix consists of four listings. The first shows the CWEB input that generated sections 12–15 of the file common.w, which contains routines common to CWEAVE and CTANGLE. Note that some of the lines are indented to show the program structure; the indentation is ignored by CWEAVE and CTANGLE, but users find that CWEB files are quite readable if they have some such indentation.

The second and third listings show corresponding parts of the C code output by CTANGLE and of the corresponding TEX code output by CWEAVE, when run on common.w. The fourth listing shows how that output looks when printed out.

```
@ Procedure |prime_the_change_buffer|
sets |change_buffer| in preparation for the next matching operation.
Since blank lines in the change file are not used for matching, we have
|(change_limit==change_buffer && !changing)| if and only if
the change file is exhausted. This procedure is called only when
|changing| is 1; hence error messages will be reported correctly.

@c
void
prime_the_change_buffer()
{
  change_limit=change_buffer; /* this value is used if the change file ends */
  @<Skip over comment lines in the change file; |return| if end of file@>;
  @<Skip to the next nonblank line; |return| if end of file@>;
  @<Move |buffer| and |limit| to |change_buffer| and |change_limit|@>;
}

@ While looking for a line that begins with \.{@@x} in the change file, we
allow lines that begin with \.{@@}, as long as they don't begin with \.{@@y},
\.{@@z} or \.{@@i} (which would probably mean that the change file is fouled up).

@<Skip over comment lines in the change file...@>=
while(1) {
  change_line++;
  if (!input_ln(change_file)) return;
  if (limit<buffer+2) continue;
  if (buffer[0]!='@@') continue;
  if (xisupper(buffer[1])) buffer[1]=tolower(buffer[1]);
  if (buffer[1]=='x') break;
  if (buffer[1]=='y' || buffer[1]=='z' || buffer[1]=='i') {
    loc=buffer+2;
    err_print("! Missing @@x in change file");
@.Missing @@x...@>
  }
}

@ Here we are looking at lines following the \.{@@x}.

@<Skip to the next nonblank line...@>=
do {
  change_line++;
  if (!input_ln(change_file)) {
    err_print("! Change file ended after @@x");
@.Change file ended...@>
    return;
  }
} while (limit==buffer);

@ @<Move |buffer| and |limit| to |change_buffer| and |change_limit|@>=
{
  change_limit=change_buffer-buffer+limit;
  strncpy(change_buffer,buffer,limit-buffer+1);
}
```

Here's the portion of the C code generated by CTANGLE that corresponds to the source on the preceding page. Notice that sections 13, 14 and 15 have been tangled into section 12.

```
/*:9*//*12:*/
#line 247 "common.w"

void
prime_the_change_buffer()
{
change_limit= change_buffer;
/*13:*/
#line 261 "common.w"

while(1){
change_line++;
if(!input_ln(change_file))return;
if(limit<buffer+2)continue;
if(buffer[0]!='@')continue;
if(xisupper(buffer[1]))buffer[1]= tolower(buffer[1]);
if(buffer[1]=='x')break;
if(buffer[1]=='y'||buffer[1]=='z'||buffer[1]=='i'){
loc= buffer+2;
err_print("! Missing @x in change file");

}
}

/*:13*/
#line 252 "common.w"
;
/*14:*/
#line 278 "common.w"

do{
change_line++;
if(!input_ln(change_file)){
err_print("! Change file ended after @x");

return;
}
}while(limit==buffer);

/*:14*/
#line 253 "common.w"
;
/*15:*/
#line 288 "common.w"

{
change_limit= change_buffer-buffer+limit;
strncpy(change_buffer,buffer,limit-buffer+1);
}

/*:15*/
#line 254 "common.w"
;
}

/*:12*//*16:*/
```

Here is the corresponding excerpt from common.tex.

```
\M{12}Procedure \PB{\\{prime\_the\_change\_buffer}}
sets \PB{\\{change\_buffer}} in preparation for the next matching operation.
Since blank lines in the change file are not used for matching, we have
\PB{$(\\{change\_limit}\E\\{change\_buffer}\W\R\\{changing})$} if and only if
the change file is exhausted. This procedure is called only when
\PB{\\{changing}} is 1; hence error messages will be reported correctly.

\Y\B\&{void} \\{prime\_the\_change\_buffer}(\,)\1\1\2\2\6
${}\{{}$\1\6
${}\\{change\_limit}\K\\{change\_buffer}{}$;\C{ this value is used if the
change file ends }\6
\X13:Skip over comment lines in the change file; \PB{\&{return}} if end of file%
\X;\6
\X14:Skip to the next nonblank line; \PB{\&{return}} if end of file\X;\6
\X15:Move \PB{\\{buffer}} and \PB{\\{limit}} to \PB{\\{change\_buffer}} and %
\PB{\\{change\_limit}}\X;\6
\4${}\}{}$\2\par
\fi

\M{13}While looking for a line that begins with \.{@x} in the change file, we
allow lines that begin with \.{@}, as long as they don't begin with \.{@y},
\.{@z} or \.{@i} (which would probably mean that the change file is fouled up).

\Y\B\4\X13:Skip over comment lines in the change file; \PB{\&{return}} if end
of file\X${}\E{}$\6
\&{while} (\T{1})\5
${}\{{}$\1\6
${}\\{change\_line}\PP;{}$\6
\&{if} ${}(\R\\{input\_ln}(\\{change\_file})){}$\1\5
\&{return};\2\6
\&{if} ${}(\\{limit}<\\{buffer}+\T{2}){}$\1\5
\&{continue};\2\6
\&{if} ${}(\\{buffer}[\T{0}]\I\.{'@'}){}$\1\5
\&{continue};\2\6
\&{if} (\\{xisupper}(\\{buffer}[\T{1}]))\1\5
${}\\{buffer}[\T{1}]\K\\{tolower}(\\{buffer}[\T{1}]);{}$\2\6
\&{if} ${}(\\{buffer}[\T{1}]\E\.{'x'}){}$\1\5
\&{break};\2\6
\&{if} ${}(\\{buffer}[\T{1}]\E\.{'y'}\V\\{buffer}[\T{1}]\E\.{'z'}\V\\{buffer}[%
\T{1}]\E\.{'i'}){}$\5
${}\{{}$\1\6
${}\\{loc}\K\\{buffer}+\T{2};{}$\6
\\{err\_print}(\.{"!\ Missing\ @x\ in\ cha}\)\.{nge\ file"});\6
\4${}\}{}$\2\6
\4${}\}{}$\2\par
\U12.\fi

\M{14}Here we are looking at lines following the \.{@x}.

\Y\B\4\X14:Skip to the next nonblank line; \PB{\&{return}} if end of file\X${}%
\E{}$\6
\&{do}\5
${}\{{}$\1\6
${}\\{change\_line}\PP;{}$\6
\&{if} ${}(\R\\{input\_ln}(\\{change\_file})){}$\5
${}\{{}$\1\6
\\{err\_print}(\.{"!\ Change\ file\ ended}\)\.{\ after\ @x"});\6
\&{return};\6
\4${}\}{}$\2\6
\4${}\}{}$\2\5
\&{while} ${}(\\{limit}\E\\{buffer}){}$;\par
\U12.\fi

\M{15}\B\X15:Move \PB{\\{buffer}} and \PB{\\{limit}} to \PB{\\{change\_buffer}}
and \PB{\\{change\_limit}}\X${}\E{}$\6
${}\{{}$\1\6
${}\\{change\_limit}\K\\{change\_buffer}-\\{buffer}+\\{limit};{}$\6
${}\\{strncpy}(\\{change\_buffer},\39\\{buffer},\39\\{limit}-\\{buffer}+%
\T{1});{}$\6
\4${}\}{}$\2\par
\Us12\ET16.\fi
```

And here's what the same excerpt looks like when typeset.

12. Procedure *prime_the_change_buffer* sets *change_buffer* in preparation for the next matching operation. Since blank lines in the change file are not used for matching, we have ($change\_limit \equiv change\_buffer \land \neg changing$) if and only if the change file is exhausted. This procedure is called only when *changing* is 1; hence error messages will be reported correctly.

```
void prime_the_change_buffer()
{
  change_limit = change_buffer;      /* this value is used if the change file ends */
  ⟨ Skip over comment lines in the change file; return if end of file 13 ⟩;
  ⟨ Skip to the next nonblank line; return if end of file 14 ⟩;
  ⟨ Move buffer and limit to change_buffer and change_limit 15 ⟩;
}
```

13. While looking for a line that begins with `@x` in the change file, we allow lines that begin with `@`, as long as they don't begin with `@y`, `@z` or `@i` (which would probably mean that the change file is fouled up).

⟨ Skip over comment lines in the change file; **return** if end of file 13 ⟩ ≡

```
while (1) {
  change_line++;
  if (¬input_ln(change_file)) return;
  if (limit < buffer + 2) continue;
  if (buffer[0] ≠ '@') continue;
  if (xisupper(buffer[1])) buffer[1] = tolower(buffer[1]);
  if (buffer[1] ≡ 'x') break;
  if (buffer[1] ≡ 'y' ∨ buffer[1] ≡ 'z' ∨ buffer[1] ≡ 'i') {
    loc = buffer + 2;
    err_print("!␣Missing␣@x␣in␣change␣file");
  }
}
```

This code is used in section 12.

14. Here we are looking at lines following the `@x`.

⟨ Skip to the next nonblank line; **return** if end of file 14 ⟩ ≡

```
do {
  change_line++;
  if (¬input_ln(change_file)) {
    err_print("!␣Change␣file␣ended␣after␣@x");
    return;
  }
} while (limit ≡ buffer);
```

This code is used in section 12.

15. ⟨ Move *buffer* and *limit* to *change_buffer* and *change_limit* 15 ⟩ ≡

```
{
  change_limit = change_buffer − buffer + limit;
  strncpy(change_buffer, buffer, limit − buffer + 1);
}
```

This code is used in sections 12 and 16.

Appendix B: The `cwebmac.tex` file

This is the file that extends "plain TeX" format in order to support the features needed by the output of CWEAVE.

```
% standard macros for CWEB listings (in addition to plain.tex)
% Version 3.0 --- June 1993
\ifx\documentstyle\undefined\else\endinput\fi % LaTeX will use other macros
\xdef\fmtversion{\fmtversion+CWEB3.0}

\let\:=\. % preserve a way to get the dot accent
 % (all other accents will still work as usual)

\parskip 0pt % no stretch between paragraphs
\parindent 1em % for paragraphs and for the first line of C text

\font\ninerm=cmr9
\let\mc=\ninerm % medium caps
\def\CEE/{{\mc C\spacefactor1000}}
\def\UNIX/{{\mc U\kern-.05emNIX\spacefactor1000}}
\def\TEX/{\TeX}
\def\CPLUSPLUS/{{\mc C\PP\spacefactor1000}}
\def\Cee{\CEE/} % for backward compatibility
\def\9#1{}
\font\eightrm=cmr8
\let\sc=\eightrm % small caps (NOT a caps-and-small-caps font)
\let\mainfont=\tenrm
\let\cmntfont\tenrm
%\font\tenss=cmss10 \let\cmntfont\tenss % alternative comment font
\font\titlefont=cmr7 scaled\magstep4 % title on the contents page
\font\ttitlefont=cmtt10 scaled\magstep2 % typewriter type in title
\font\tentex=cmtex10 % TeX extended character set (used in strings)
\fontdimen7\tentex=0pt % no double space after sentences

\def\\#1{\leavevmode\hbox{\it#1\/\kern.05em}} % italic type for identifiers
\def\|#1{\leavevmode\hbox{$#1$}} % one-letter identifiers look better this way
\def\&#1{\leavevmode\hbox{\bf
  \def\_{\kern.04em\vbox{\hrule width.3em height .6pt}\kern.08em}%
  #1\/\kern.05em}} % boldface type for reserved words
\def\.#1{\leavevmode\hbox{\tentex % typewriter type for strings
  \let\\=\BS % backslash in a string
  \let\{=\LB % left brace in a string
  \let\}=\RB % right brace in a string
  \let\~=\TL % tilde in a string
  \let\ =\SP % space in a string
  \let\_=\UL % underline in a string
  \let\&=\AM % ampersand in a string
  \let\^=\CF % circumflex in a string
  #1\kern.05em}}
\def\){\discretionary{\hbox{\tentex\BS}}{}{}}
\def\AT{@} % at sign for control text (not needed in versions >= 2.9)
\def\ATL{\par\noindent\bgroup\catcode`\_=12 \postATL} % print @l in limbo
\def\postATL#1 #2 {\bf letter \\{\uppercase{\char"#1}}
   tangles as \tentex "#2"\egroup\par}
\def\noATL#1 #2 {}
\def\noatl{\let\ATL=\noATL} % suppress output from @l
\def\ATH{\X\kern-.5em:Preprocessor definitions\X}
\let\PB=\relax % hook for program brackets |...| in TeX part or section name

\chardef\AM=`\& % ampersand character in a string
\chardef\BS=`\\ % backslash in a string
\chardef\LB=`\{ % left brace in a string
\chardef\RB=`\} % right brace in a string
\def\SP{{\tt\char`\ }} % (visible) space in a string
\chardef\TL=`\~ % tilde in a string
\chardef\UL=`\_ % underline character in a string
\chardef\CF=`\^ % circumflex character in a string

\newbox\PPbox % symbol for ++
\setbox\PPbox=\hbox{\kern.5pt\raise1pt\hbox{\sevenrm+\kern-1pt+}\kern.5pt}
\def\PP{\copy\PPbox}
```

```
\newbox\MMbox \setbox\MMbox=\hbox{\kern.5pt\raise1pt\hbox{\sevensy\char0
 \kern-1pt\char0}\kern.5pt}
\def\MM{\copy\MMbox}
\newbox\MGbox % symbol for ->
\setbox\MGbox=\hbox{\kern-2pt\lower3pt\hbox{\teni\char'176}\kern1pt}
\def\MG{\copy\MGbox}
\def\MRL#1{\mathrel{\let\K==#1}}
%\def\MRL#1{\KK#1}\def\KK#1#2{\buildrel\;#1\over{#2}}
\let\GG=\gg
\let\LL=\ll
\let\NULL=\Lambda
\mathchardef\AND="2026 % bitwise and; also \& (unary operator)
\let\OR=\mid % bitwise or
\let\XOR=\oplus % bitwise exclusive or
\def\CM{{\sim}} % bitwise complement
\newbox\MODbox \setbox\MODbox=\hbox{\eightrm\%}
\def\MOD{\mathbin{\copy\MODbox}}
\def\DC{\kern.1em{::}\kern.1em} % symbol for ::
\def\PA{\mathbin{.*}} % symbol for .*
\def\MGA{\mathbin{\MG*}} % symbol for ->*
\def\this{\&{this}}

\newbox\bak \setbox\bak=\hbox to -1em{} % backspace one em
\newbox\bakk\setbox\bakk=\hbox to -2em{} % backspace two ems

\newcount\ind % current indentation in ems
\def\1{\global\advance\ind by1\hangindent\ind em} % indent one more notch
\def\2{\global\advance\ind by-1} % indent one less notch
\def\3#1{\hfil\penalty#10\hfilneg} % optional break within a statement
\def\4{\copy\bak} % backspace one notch
\def\5{\hfil\penalty-1\hfilneg\kern2.5em\copy\bakk\ignorespaces}% optional break
\def\6{\ifmmode\else\par % forced break
  \hangindent\ind em\noindent\kern\ind em\copy\bakk\ignorespaces\fi}
\def\7{\Y\6} % forced break and a little extra space
\def\8{\hskip-\ind em\hskip 2em} % no indentation

\newcount\gdepth % depth of current major group, plus one
\newcount\secpagedepth
\secpagedepth=3 % page breaks will occur for depths -1, 0, and 1
\newtoks\gtitle % title of current major group
\newskip\intersecskip \intersecskip=12pt minus 3pt % space between sections
\let\yskip=\smallskip
\def\?{\mathrel?}
\def\note#1#2.{\Y\noindent{\hangindent2em\baselineskip10pt\eightrm#1~#2.\par}}
\def\lapstar{\rlap{*}}
\def\stsec{\rightskip=0pt % get out of C mode (cf. \B)
  \sfcode`;=1500 \pretolerance 200 \hyphenpenalty 50 \exhyphenpenalty 50
  \noindent{\let\*=\lapstar\bf\secstar.\quad}}
\let\startsection=\stsec
\def\defin#1{\global\advance\ind by 2 \1\&{#1 } } % begin `define' or `format'
\def\A{\note{See also section}} % xref for doubly defined section name
\def\As{\note{See also sections}} % xref for multiply defined section name
\def\B{\rightskip=0pt plus 100pt minus 10pt % go into C mode
  \sfcode`;=3000
  \pretolerance 10000
  \hyphenpenalty 9999 % so strings can be broken (discretionary \ is inserted)
  \exhyphenpenalty 10000
  \global\ind=2 \1\ \unskip}
\def\C#1{\5\5\quad$/\ast\,${\cmntfont #1}$\,\ast/$}
\let\SHC\C % "// short comments" treated like "/* ordinary comments */"
%\def\C#1{\5\5\quad$\triangleright\,${\cmntfont#1}$\,\triangleleft$}
%\def\SHC#1{\5\5\quad$\diamond\,${\cmntfont#1}}
\def\D{\defin{\#define}} % macro definition
\let\E=\equiv % equivalence sign
\def\ET{ and~} % conjunction between two section numbers
\def\ETs{, and~} % conjunction between the last two of several section numbers
\def\F{\defin{format}} % format definition
\let\G=\ge % greater than or equal sign
% \H is long Hungarian umlaut accent
\let\I=\ne % unequal sign
\def\J{\.{@\&}} % TANGLE's join operation
```

```
\let\K== % assignment operator
%\let\K=\leftarrow % "honest" alternative to standard assignment operator
% \L is Polish letter suppressed-L
\outer\def\M#1{\MN{#1}\ifon\vfil\penalty-100\vfilneg % beginning of section
  \vskip\intersecskip\startsection\ignorespaces}
\outer\def\N#1#2#3.{\gdepth=#1\gtitle={#3}\MN{#2}% beginning of starred section
  \ifon\ifnum#1<\secpagedepth \vfil\eject % force page break if depth is small
    \else\vfil\penalty-100\vfilneg\vskip\intersecskip\fi\fi
  \message{*\secno} % progress report
  \edef\next{\write\cont{\ZZ{#3}{#1}{\secno}% write to contents file
                  {\noexpand\the\pageno}}}\next % \ZZ{title}{depth}{sec}{page}
  \ifon\startsection{\bf#3.\quad}\ignorespaces}
\def\MN#1{\par % common code for \M, \N
  {\xdef\secstar{#1}\let\*=\empty\xdef\secno{#1}}% remove \* from section name
  \ifx\secno\secstar \onmaybe \else\ontrue \fi
  \mark{{{\tensy x}\secno}{\the\gdepth}{\the\gtitle}}}
% each \mark is {section reference or null}{depth plus 1}{group title}
% \O is Scandinavian letter O-with-slash
% \P is paragraph sign
\def\Q{\note{This code is cited in section}} % xref for mention of a section
\def\Qs{\note{This code is cited in sections}} % xref for mentions of a section
\let\R=\lnot % logical not
% \S is section sign
\def\T#1{\leavevmode % octal, hex or decimal constant
  \hbox{$\def\?{\kern.2em}%
    \def\$##1{\egroup_{\,\rm##1}\bgroup}% suffix to constant
    \def\_{\cdot 10^{\aftergroup}}% power of ten (via dirty trick)
    \let\~=\oct \let\^=\hex {#1}$}}
\def\U{\note{This code is used in section}} % xref for use of a section
\def\Us{\note{This code is used in sections}} % xref for uses of a section
\let\V=\lor % logical or
\let\W=\land % logical and
\def\X#1:#2\X{\ifmmode\gdef\XX{\null$\null}\else\gdef\XX{}\fi % section name
  \XX$\langle\,${#2\eightrm\kern.5em#1}$\,\rangle$\XX}
\def\Y{\par\yskip}
\let\Z=\le
\let\ZZ=\let % now you can \write the control sequence \ZZ
\let\*=*

%\def\oct{\hbox{\rm\char'23\kern-.2em\it\aftergroup\?\aftergroup}} % WEB style
%\def\hex{\hbox{\rm\char"7D\tt\aftergroup}} % WEB style
\def\oct{\hbox{$^\circ$\kern-.1em\it\aftergroup\?\aftergroup}}% CWEB style
\def\hex{\hbox{$^{\scriptscriptstyle\#}$\tt\aftergroup}} % CWEB style
\def\vb#1{\leavevmode\hbox{\kern2pt\vrule\vtop{\vbox{\hrule
        \hbox{\strut\kern2pt\.{#1}\kern2pt}}
      \hrule}\vrule\kern2pt}} % verbatim string

\def\onmaybe{\let\ifon=\maybe} \let\maybe=\iftrue
\newif\ifon \newif\iftitle \newif\ifpagesaved

\def\lheader{\mainfont\the\pageno\eightrm\qquad\grouptitle\hfill\title\qquad
  \mainfont\topsecno} % top line on left-hand pages
\def\rheader{\mainfont\topsecno\eightrm\qquad\title\hfill\grouptitle
  \qquad\mainfont\the\pageno} % top line on right-hand pages
\def\grouptitle{\let\i=I\let\j=J\uppercase\expandafter{\expandafter
                        \takethree\topmark}}
\def\topsecno{\expandafter\takeone\topmark}
\def\takeone#1#2#3{#1}
\def\taketwo#1#2#3{#2}
\def\takethree#1#2#3{#3}
\def\nullsec{\eightrm\kern-2em} % the \kern-2em cancels \qquad in headers

\let\page=\pagebody \raggedbottom
% \def\page{\box255 }\normalbottom % faster, but loses plain TeX footnotes
\def\normaloutput#1#2#3{\shipout\vbox{
  \ifodd\pageno\hoffset=\pageshift\fi
  \vbox to\fullpageheight{
  \iftitle\global\titlefalse
  \else\hbox to\pagewidth{\vbox to10pt{}\ifodd\pageno #3\else#2\fi}\fi
  \vfill#1}} % parameter #1 is the page itself
  \global\advance\pageno by1}
```

```
\gtitle={\.{CWEB} output} % this running head is reset by starred sections
\mark{\noexpand\nullsec0{\the\gtitle}}
\def\title{\expandafter\uppercase\expandafter{\jobname}}
\def\topofcontents{\centerline{\titlefont\title}\vskip.7in
  \vfill} % this material will start the table of contents page
\def\botofcontents{\vfill
  \centerline{\covernote}} % this material will end the table of contents page
\def\covernote{}
\def\contentspagenumber{0} % default page number for table of contents
\newdimen\pagewidth \pagewidth=6.5in % the width of each page
\newdimen\pageheight \pageheight=8.7in % the height of each page
\newdimen\fullpageheight \fullpageheight=9in % page height including headlines
\newdimen\pageshift \pageshift=0in % shift righthand pages wrt lefthand ones
\def\magnify#1{\mag=#1\pagewidth=6.5truein\pageheight=8.7truein
  \fullpageheight=9truein\setpage}
\def\setpage{\hsize\pagewidth\vsize\pageheight} % use after changing page size
\def\contentsfile{\jobname.toc} % file that gets table of contents info
\def\readcontents{\input \contentsfile}
\def\readindex{\input \jobname.idx}
\def\readsections{\input \jobname.scn}

\newwrite\cont
\output{\setbox0=\page % the first page is garbage
  \openout\cont=\contentsfile
       \write\cont{\catcode `\noexpand\@=11\relax}   % \makeatletter
  \global\output{\normaloutput\page\lheader\rheader}}
\setpage
\vbox to \vsize{} % the first \topmark won't be null

\def\ch{\note{The following sections were changed by the change file:}
  \let\*=\relax}
\newbox\sbox % saved box preceding the index
\newbox\lbox % lefthand column in the index
\def\inx{\par\vskip6pt plus 1fil % we are beginning the index
  \def\page{\box255 } \normalbottom
  \write\cont{} % ensure that the contents file isn't empty
       \write\cont{\catcode `\noexpand\@=12\relax}   % \makeatother
  \closeout\cont % the contents information has been fully gathered
  \output{\ifpagesaved\normaloutput{\box\sbox}\lheader\rheader\fi
    \global\setbox\sbox=\page \global\pagesavedtrue}
  \pagesavedfalse \eject % eject the page-so-far and predecessors
  \setbox\sbox\vbox{\unvbox\sbox} % take it out of its box
  \vsize=\pageheight \advance\vsize by -\ht\sbox % the remaining height
  \hsize=.5\pagewidth \advance\hsize by -10pt
    % column width for the index (20pt between cols)
  \parfillskip 0pt plus .6\hsize % try to avoid almost empty lines
  \def\lr{L} % this tells whether the left or right column is next
  \output{\if L\lr\global\setbox\lbox=\page \gdef\lr{R}
    \else\normaloutput{\vbox to\pageheight{\box\sbox\vss
        \hbox to\pagewidth{\box\lbox\hfil\page}}}\lheader\rheader
    \global\vsize\pageheight\gdef\lr{L}\global\pagesavedfalse\fi}
  \message{Index:}
  \parskip 0pt plus .5pt
  \outer\def\I##1, {\par\hangindent2em\noindent##1:\kern1em} % index entry
  \def\[##1]{$\underline{##1}$} % underlined index item
  \rm \rightskip0pt plus 2.5em \tolerance 10000 \let\*=\lapstar
  \hyphenpenalty 10000 \parindent0pt
  \readindex}
\def\fin{\par\vfill\eject % this is done when we are ending the index
  \ifpagesaved\null\vfill\eject\fi % output a null index column
  \if L\lr\else\null\vfill\eject\fi % finish the current page
  \parfillskip 0pt plus 1fil
  \def\grouptitle{NAMES OF THE SECTIONS}
  \let\topsecno=\nullsec
  \message{Section names:}
  \output={\normaloutput\page\lheader\rheader}
  \setpage
  \def\note##1##2.{\quad{\eightrm##1~##2.}}
  \def\Q{\note{Cited in section}} % crossref for mention of a section
  \def\Qs{\note{Cited in sections}} % crossref for mentions of a section
```

```
  \def\U{\note{Used in section}} % crossref for use of a section
  \def\Us{\note{Used in sections}} % crossref for uses of a section
  \def\I{\par\hangindent 2em}\let\*=*
  \readsections}
\def\con{\par\vfill\eject % finish the section names
% \ifodd\pageno\else\titletrue\null\vfill\eject\fi % for duplex printers
  \rightskip 0pt \hyphenpenalty 50 \tolerance 200
  \setpage \output={\normaloutput\page\lheader\rheader}
  \titletrue % prepare to output the table of contents
  \pageno=\contentspagenumber
  \def\grouptitle{TABLE OF CONTENTS}
  \message{Table of contents:}
  \topofcontents
  \line{\hfil Section\hbox to3em{\hss Page}}
  \let\ZZ=\contentsline
  \readcontents\relax % read the contents info
  \botofcontents \end} % print the contents page(s) and terminate
\def\contentsline#1#2#3#4{\ifnum#2=0 \smallbreak\fi
    \line{\consetup{#2}#1
      \rm\leaders\hbox to .5em{.\hfil}\hfil\ #3\hbox to3em{\hss#4}}}
\def\consetup#1{\ifcase#1 \bf % depth -1 (@**)
  \or % depth 0 (@*)
  \or \hskip2em % depth 1 (@*1)
  \or \hskip4em % depth 2 (@*2)
  \or \hskip6em % depth 3 (@*3)
  \or \hskip8em % depth 4 (@*4)
  \or \hskip10em % depth 5 (@*5)
  \else \hskip12em \fi} % depth 6 or more
\def\noinx{\let\inx=\end} % no indexes or table of contents
\def\nosecs{\let\FIN=\fin \def\fin{\let\parfillskip=\end \FIN}}
    % no index of section names or table of contents
\def\nocon{\let\con=\end} % no table of contents
\def\today{\ifcase\month\or
  January\or February\or March\or April\or May\or June\or
  July\or August\or September\or October\or November\or December\fi
  \space\number\day, \number\year}
\newcount\twodigits
\def\hours{\twodigits=\time \divide\twodigits by 60 \printtwodigits
  \multiply\twodigits by-60 \advance\twodigits by\time :\printtwodigits}
\def\gobbleone1{}
\def\printtwodigits{\advance\twodigits100
  \expandafter\gobbleone\number\twodigits
  \advance\twodigits-100 }
\def\TeX{{\ifmmode\it\fi
   \leavevmode\hbox{T\kern-.1667em\lower.424ex\hbox{E}\hskip-.125em X}}}
\def\,{\relax\ifmmode\mskip\thinmuskip\else\thinspace\fi}
\def\datethis{\def\startsection{\leftline{\sc\today\ at \hours}\bigskip
  \let\startsection=\stsec\stsec}}
  % say `\datethis' in limbo, to get your listing timestamped before section 1
\def\datecontentspage{%
  \def\topofcontents{\leftline{\sc\today\ at \hours}\bigskip
   \centerline{\titlefont\title}\vfill}} % timestamps the contents page
```

Appendix C: How to use CWEB macros

The macros in cwebmac make it possible to produce a variety of formats without editing the output of CWEAVE, and the purpose of this appendix is to explain some of the possibilities.

1. Four fonts have been declared in addition to the standard fonts of PLAIN format: You can say '{\mc UNIX}' to get UNIX in medium-size caps; you can say '{\sc STUFF}' to get STUFF in small caps; and you can select the largish fonts \titlefont and \ttitlefont in the title of your document, where \ttitlefont is a typewriter style of type. There are macros \UNIX/ and \CEE/ to refer to UNIX and C with medium-size caps.

2. When you mention an identifier in TEX text, you normally call it '|identifier|'. But you can also say '\\{identifier}'. The output will look the same in both cases, but the second alternative doesn't put *identifier* into the index, since it bypasses CWEAVE's translation from C mode. In the second case you have to put a backslash before each underline character in the identifier.

3. To get typewriter-like type, as when referring to 'CWEB', you can use the '\.' macro (e.g., '\.{CWEB}'). In the argument to this macro you should insert an additional backslash before the symbols listed as 'special string characters' in the index to CWEAVE, i.e., before backslashes and dollar signs and the like. A '\␣' here will result in the visible space symbol; to get an invisible space following a control sequence you can say '{␣}'. If the string is long, you can break it up into substrings that are separated by '\)'; the latter gives a discretionary backslash if TEX has to break a line here.

4. The three control sequences \pagewidth, \pageheight, and \fullpageheight can be redefined in the limbo section at the beginning of your CWEB file, to change the dimensions of each page. The default settings

```
\pagewidth=6.5in
\pageheight=8.7in
\fullpageheight=9in
```

were used to prepare this manual; \fullpageheight is \pageheight plus room for the additional heading and page numbers at the top of each page. If you change any of these quantities, you should call the macro \setpage immediately after making the change.

5. The \pageshift macro defines an amount by which right-hand pages (i.e., odd-numbered pages) are shifted right with respect to left-hand (even-numbered) ones. By adjusting this amount you may be able to get two-sided output in which the page numbers line up on opposite sides of each sheet.

6. The \title macro will appear at the top of each page in small caps; it is the job name unless redefined.

7. The first page usually is assigned page number 1. To start on page 16, with contents on page 15, say this: '\def\contentspagenumber{15} \pageno=\contentspagenumber \advance\pageno by 1'.

8. The macro \iftitle will suppress the header line if it is defined by '\titletrue'. The normal value is \titlefalse except for the table of contents; thus, the contents page is usually unnumbered.

Two macros are provided to give flexibility to the table of contents: \topofcontents is invoked just before the contents info is read, and \botofcontents is invoked just after. Here's a typical definition:

```
\def\topofcontents{\null\vfill
  \titlefalse % include headline on the contents page
  \def\rheader{\mainfont The {\tt CWEAVE} processor\hfil}
  \centerline{\titlefont The {\ttitlefont CWEAVE} processor}
  \vskip 15pt \centerline{(Version 3.0)} \vfill}
```

Redefining \rheader, which is the headline for right-hand pages, suffices in this case to put the desired information at the top of the contents page.

9. Data for the table of contents is written to a file that is read after the indexes have been TEXed; there's one line of data for every starred section. The file common.toc might look like this:

```
\ZZ {Introduction}{0}{1}{23}
\ZZ {The character set}{2}{5}{24}
```

and so on. The \topofcontents macro could redefine \ZZ so that the information appears in any desired format.

10. Sometimes it is necessary or desirable to divide the output of `CWEAVE` into subfiles that can be processed separately. For example, the listing of TeX runs to more than 500 pages, and that is enough to exceed the capacity of many printing devices and/or their software. When an extremely large job isn't cut into smaller pieces, the entire process might be spoiled by a single error of some sort, making it necessary to start everything over.

Here's a safe way to break a woven file into three parts: Say the pieces are α, β, and γ, where each piece begins with a starred section. All macros should be defined in the opening limbo section of α, and copies of this TeX code should be placed at the beginning of β and of γ. In order to process the parts separately, we need to take care of two things: The starting page numbers of β and γ need to be set up properly, and the table of contents data from all three runs needs to be accumulated.

The `cwebmac` macros include two control sequences `\contentsfile` and `\readcontents` that facilitate the necessary processing. We include '`\def\contentsfile{cont1}`' in the limbo section of α, and we include '`\def\contentsfile{cont2}`' in the limbo section of β; this causes TeX to write the contents data for α and β into `cont1.tex` and `cont2.tex`. Now in γ we say

```
\def\readcontents{\input cont1 \input cont2 \input \contentsfile};
```

this brings in the data from all three pieces, in the proper order.

However, we still need to solve the page-numbering problem. One way to do it is to include the following in the limbo material for β:

```
\message{Please type the last page number of part 1: }
\read -1 to \temp \pageno=\temp \advance\pageno by 1
```

Then you simply provide the necessary data when TeX requests it; a similar construction is used at the beginning of γ.

This method can, of course, be used to divide a woven file into any number of pieces.

11. Sometimes it is nice to include things in the index that are typeset in a special way. For example, we might want to have an index entry for 'TeX'. `CWEAVE` provides two simple ways to typeset an index entry (unless the entry is an identifier or a reserved word): '`@^`' gives roman type, and '`@.`' gives typewriter type. But if we try to typeset 'TeX' in roman type by saying, e.g., '`@^\TeX@>`', the backslash character gets in the way, and this entry wouldn't appear in the index with the T's.

The solution is to use the '`@:`' feature, declaring a macro that simply removes a sort key as follows:

```
\def\9#1{}
```

Now you can say, e.g., '`@:TeX}{\TeX@>`' in your `CWEB` file; `CWEAVE` puts it into the index alphabetically, based on the sort key, and produces the macro call '`\9{TeX}{\TeX}`' which will ensure that the sort key isn't printed.

A similar idea can be used to insert hidden material into section names so that they are alphabetized in whatever way you might wish. Some people call these tricks "special refinements"; others call them "kludges".

12. The control sequence `\secno` is set to the number of the section being typeset.

13. If you want to list only the sections that have changed, together with the index, put the command '`\let\maybe=\iffalse`' in the limbo section before the first section of your `CWEB` file. It's customary to make this the first change in your change file.

14. To get output in languages other than English, redefine the macros `\A`, `\ET`, `\Q`, `\U`, `\ch`, `\fin`, `\con`, `\today`, `\datethis`, and `\datecontentspage`. `CWEAVE` itself need not be changed.

15. Some output can be selectively suppressed with the macros `\noatl`, `\noinx`, `\nosecs`, `\nocon`.

16. All accents and special text symbols of plain TeX format will work in `CWEB` documents just as they are described in Chapter 9 of *The TeXbook*, with one exception. The dot accent (normally `\.`) must be typed `\:` instead.

17. Several commented-out lines in `cwebmac.tex` are suggestions that users may wish to adopt. For example, one such line inserts a blank page if you have a duplex printer. Appendices D, E, and F of the complete version of this manual are printed using a commented-out option that substitutes '$\leftarrow$' for '=' in the program listings. Looking at those appendices might help you decide which format you like better.