The WEB System of Structured Documentation

This memo describes how to write programs in the WEB language; and it also includes the full WEB documentation for WEAVE and TANGLE, the programs that read WEB input and produce TEX and Pascal output, respectively. The philosophy behind WEB is that an experienced system programmer, who wants to provide the best possible documentation of software products, needs two things simultaneously: a language like TEX for formatting, and a language like Pascal 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 Pascal 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.

Since WEB is an experimental system developed for internal use within the TEX project at Stanford, this report is rather terse, and it assumes that the reader is an experienced programmer who is highly motivated to read a detailed description of WEB's rules. Furthermore, even if a less terse manual were to be written, the reader would have to be warned in advance that WEB is not for beginners and it never will be: The user of WEB must be familiar with both TEX and Pascal. When one writes a WEB description of a software system, it is possible to make mistakes by breaking the rules of WEB and/or the rules of TEX and/or the rules of Pascal. In practice, all three types of errors will occur, and you will get different error messages from the different language processors. In compensation for the sophisticated expertise needed to cope with such a variety of languages, however, experience has shown that reliable software can be created quite rapidly by working entirely in WEB from the beginning; and the documentation of such programs seems to be better than the documentation obtained by any other known method. Thus, WEB users need to be highly qualified, but they can get some satisfaction and perhaps even a special feeling of accomplishment when they have successfully created a software system with this method.

To use WEB, you prepare a file called COB.WEB (say), and then you apply a system program called WEAVE to this file, obtaining an output file called COB.TEX. When TEX processes COB.TEX, your output will be a "pretty printed" version of COB.WEB that takes appropriate care of typographic details like page layout and the use of indentation, italics, boldface, etc.; this output will contain extensive cross-index information that is gathered automatically. You can also submit the same file COB.WEB to another system program called TANGLE, which will produce a file COB.PAS that contains the Pascal code of your COB program. The Pascal compiler will convert COB.PAS into machine-language instructions corresponding to the algorithms that were so nicely formatted by WEAVE and TEX. Finally, you can (and should) delete the files COB.TEX and COB.PAS, because COB.WEB contains the definitive source code. Examples of the behavior of WEAVE and TANGLE are appended to this manual.

Besides providing a documentation tool, WEB enhances the Pascal language by providing a rudimentary macro capability together with the ability to permute pieces of the program text, so that a large system can be understood entirely in terms of small modules and their local interrelationships. The TANGLE program is so named because it takes a given web and moves the modules from their web structure into the order required by Pascal; the advantage of programming in WEB is that the algorithms can be expressed in "untangled" form, with each module explained separately. The WEAVE program is so named because it takes a given web and intertwines the TEX and Pascal portions contained in each module, then it knits the whole fabric into a structured document. (Get it? Wow.) Perhaps there is some deep connection here with the fact that the German word for "weave" is "*web*", and the corresponding Latin imperative is "*texe*"!

It is impossible to list all of the related work that has influenced the design of WEB, but the key contributions should be mentioned here. (1) Myrtle Kellington, as executive editor for ACM publications, developed excellent typographic standards for the typesetting of Algol programs during the 1960s, based on the original

designs of Peter Naur; the subtlety and quality of this influential work can be appreciated only by people who have seen what happens when other printers try to typeset Algol without the advice of ACM's copy editors. (2) Bill McKeeman introduced a program intended to automate some of this task [Algorithm 268, "Algol 60 reference language editor," *CACM* **8** (1965), 667–668]; and a considerable flowering of such programs has occurred in recent years [see especially Derek Oppen, "Prettyprinting," *ACM TOPLAS* **2** (1980), 465–483; G. A. Rose and J. Welsh, "Formatted programming languages," *SOFTWARE Practice & Exper.* **11** (1981), 651–669]. (3) The top-down style of exposition encouraged by WEB was of course chiefly influenced by Edsger Dijkstra's essays on structured programming in the late 1960s. The less well known work of Pierre-Arnoul de Marneffe ["Holon programming: A survey," Univ. de Liege, Service Informatique, Liege, Belgium, 1973; 135 pp.] also had a significant influence on the author as WEB was being formulated. (4) Edwin Towster has proposed a similar style of documentation in which the programmer is supposed to specify the relevant data structure environment in the name of each submodule ["A convention for explicit declaration of environments and top-down refinement of data," *IEEE Trans. on Software Eng.* **SE–5** (1979), 374–386]; this requirement seems to make the documentation a bit too verbose, although experience with WEB has shown that any unusual control structure or data structure should definitely be incorporated into the module names on psychological grounds. (5) Discussions with Luis Trabb Pardo in the spring of 1979 were extremely helpful for setting up a prototype version of WEB that was called DOC. (6) Ignacio Zabala's extensive experience with DOC, in which he created a full implementation of TEX in Pascal that was successfully transported to many different computers, was of immense value while WEB was taking its present form. (7) David R. Fuchs made several crucial suggestions about how to make WEB more portable; he and Arthur L. Samuel coordinated the initial installations of WEB on dozens of computer systems, making changes to the code so that it would be acceptable to a wide variety of Pascal compilers. (8) The name WEB itself was chosen in honor of my wife's mother, Wilda Ernestine Bates.

The appendices to this report contain complete WEB programs for the WEAVE and TANGLE processors. A study of these examples, together with an attempt to write WEB programs by yourself, is the best way to understand why WEB has come to be like it is.

General rules. A WEB file is a long string of text that has been divided into individual lines. The exact line boundaries are not terribly crucial, and a programmer can pretty much chop up the WEB file in whatever way seems to look best as the file is being edited; but string constants and control texts must end on the same line on which they begin, since this convention helps to keep errors from propagating. The end of a line means the same thing as a blank space.

Two kinds of material go into WEB files: TEX text and Pascal text. A programmer writing in WEB should be thinking both of the documentation and of the Pascal program that he or she is creating; i.e., the programmer should be instinctively aware of the different actions that WEAVE and TANGLE will perform on the WEB file. TEX text is essentially copied without change by WEAVE, and it is entirely deleted by TANGLE, since the TEX text is "pure documentation." Pascal text, on the other hand, is formatted by WEAVE and it is shuffled around by TANGLE, 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 WEB programs is something like writing TEX documents, but with an additional "Pascal mode" that is added to TEX's horizontal mode, vertical mode, and math mode.

A WEB file is built up from units called *modules* that are more or less self-contained. Each module has three parts:

1) A TEX part, containing explanatory material about what is going on in the module.
2) A definition part, containing macro definitions that serve as abbreviations for Pascal constructions that would be less comprehensible if written out in full each time.
3) A Pascal part, containing a piece of the program that TANGLE will produce. This Pascal 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 module must appear in this order; i.e., the TEX commentary must come first, then the definitions, and finally the Pascal code. Any of the parts may be empty.

A module begins with the pair of symbols '@␣' or '@*', where '␣' denotes a blank space. A module ends at the beginning of the next module (i.e., at the next '@␣' or '@*'), or at the end of the file, whichever comes first. The WEB file may also contain material that is not part of any module at all, namely the text (if any) that occurs before the first module. Such text is said to be "in limbo"; it is ignored by TANGLE and copied essentially verbatim by WEAVE, 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 WEB file with TEX code in limbo that loads special fonts, defines special macros, changes the page sizes, and/or produces a title page.

Modules are numbered consecutively, starting with 1; these numbers appear at the beginning of each module of the TEX documentation, and they appear as bracketed comments at the beginning of the code generated by that module in the Pascal program.

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

For expository purposes, a module name should be a good description of the contents of that module; i.e., it should stand for the abstraction represented by the module. Then the module can be "plugged into" one or more other modules in such a way that unimportant details of its inner workings are suppressed. A module name therefore ought to be long enough to convey the necessary meaning. Unfortunately, however, 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 WEAVE and TANGLE will be able to match the names to the modules. To ameliorate this difficulty, WEAVE and TANGLE let you abbreviate a module name after its first appearance in the WEB file; you can type simply '@<α...@>', where α is any string that is a prefix of exactly one module name appearing in the file. For example, '@<Clear the arrays@>' can be abbreviated to '@<Clear...@>' if no other module name begins with the five letters 'Clear'. Module names must otherwise match character for character, except that consecutive blank spaces and/or tab marks are treated as equivalent to single spaces, 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.

We have said that a module begins with '@␣' or '@*', but we didn't say how it gets divided up into a TEX part, a definition part, and a Pascal part. The definition part begins with the first appearance of '@d' or '@f' in the module, and the Pascal part begins with the first appearance of '@p' or '@<'. The latter option '@<' stands for the beginning of a module name, which is the name of the module itself. An equals sign (=) must follow the '@>' at the end of this module name; you are saying, in effect, that the module name stands for the Pascal text that follows, so you say '⟨ module name ⟩ = Pascal text'. Alternatively, if the Pascal part begins with '@p' instead of a module name, the current module is said to be *unnamed*. Note that module names cannot appear in the definition part of a module, because the first '@<' in a module signals the beginning of its Pascal part. But any number of module names might appear in the Pascal part, once it has started.

The general idea of TANGLE is to make a Pascal program out of these modules in the following way: First all the Pascal parts of unnamed modules are copied down, in order; this constitutes the initial approximation T_0 to the text of the program. (There should be at least one unnamed module, otherwise there will be no program.) Then all module names that appear in the initial text T_0 are replaced by the Pascal parts of the corresponding modules, and this substitution process continues until no module names remain. Then all defined macros are replaced by their equivalents, according to certain rules that are explained later. The resulting Pascal code is "sanitized" so that it will be acceptable to an average garden-variety Pascal compiler; i.e., lowercase letters are converted to uppercase, long identifiers are chopped, and the lines of the output file are constrained to be at most 72 characters long. All comments will have been removed from this Pascal program except for the meta-comments delimited by '@{' and '@}', as explained below, and except for the module-number comments that point to the source location where each piece of the program text originated in the WEB file.

If the same name has been given to more than one module, the Pascal text for that name is obtained by putting together all of the Pascal parts in the corresponding modules. This feature is useful, for example, in

a module named 'Global variables in the outer block', since one can then declare global variables in whatever modules those variables are introduced. When several modules have the same name, WEAVE assigns the first module number as the number corresponding to that name, and it inserts a note at the bottom of that module telling the reader to 'See also sections so-and-so'; this footnote gives the numbers of all the other modules having the same name as the present one. The Pascal text corresponding to a module is usually formatted by WEAVE so that the output has an equivalence sign in place of the equals sign in the WEB file; i.e., the output says '⟨ module name ⟩ ≡ Pascal text'. However, in the case of the second and subsequent appearances of a module with the same name, this '≡' sign is replaced by '+≡', as an indication that the Pascal text that follows is being appended to the Pascal text of another module.

The general idea of WEAVE is to make a TEX file from the WEB file in the following way: The first line of the TEX file will be '`\input webmac`'; this will cause TEX to read in the macros that define WEB's documentation conventions. The next lines of the file will be copied from whatever TEX text is in limbo before the first module. Then comes the output for each module in turn, possibly interspersed with end-of-page marks. Finally, WEAVE will generate a cross-reference index that lists each module number in which each Pascal identifier appears, and it will also generate an alphabetized list of the module names, as well as a table of contents that shows the page and module numbers for each "starred" module.

What is a "starred" module, you ask? A module that begins with '`@*`' instead of '`@␣`' is slightly special in that it denotes a new major group of modules. The '`@*`' should be followed by the title of this group, followed by a period. Such modules 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 module. The title will also appear in the table of contents, and in boldface type at the beginning of its module. Caution: Do not use TEX control sequences in such titles, unless you know that the webmac 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 modules, 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 WEAVE for each module consists of the following: First comes the module number (e.g., '`\M123.`' at the beginning of module 123, except that '`\N`' appears in place of '`\M`' at the beginning of a starred module). Then comes the TEX part of the module, copied almost verbatim except as noted below. Then comes the definition part and the Pascal part, formatted so that there will be a little extra space between them if both are nonempty. The definition and Pascal parts are obtained by inserting a bunch of funny looking TEX macros into the Pascal program; these macros handle typographic details about fonts and proper math spacing, as well as line breaks and indentation.

When you are typing TEX text, you will probably want to make frequent reference to variables and other quantities in your Pascal 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 WEB language allows you to get the effect of Pascal editing within TEX text, if you place '|' marks before and after the Pascal material. For example, suppose you want to say something like this:

The characters are placed into *buffer*, which is a **packed array** [1 .. *n*] **of** *char*.

The TEX text would look like this in your WEB file:

```
The characters are placed into |buffer|, which is a |packed array [1..n] of char|.
```

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

```
The characters are placed into \\{buffer},
which is a \&{packed} \&{array} $ [1\to\|n]$ \&{of} \\{char}.
```

Incidentally, the cross-reference index that WEAVE would make, in the presence of a comment like this, would include the current module number as one of the index entries for *buffer* and *char*, even though *buffer* and

char might not appear in the Pascal part of this module. 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 **packed** and **array** and n and **of** would not be indexed, because WEAVE 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.

Speaking of identifiers, the author of WEB thinks that *IdentifiersSeveralWordsLong* look terribly ugly when they mix uppercase and lowercase letters. He recommends that *identifiers_several_words_long* be written with underline characters to get a much better effect. The actual identifiers sent to the Pascal compiler by TANGLE will have such underlines removed, and TANGLE will check to make sure that two different identifiers do not become identical when this happens. (In fact, TANGLE even checks that the first seven characters of identifiers are unique, when lowercase letters have been converted to uppercase; the number seven in this constraint is more strict than Pascal's eight, and it can be changed if desired.) The WEAVE processor will properly alphabetize identifiers that have embedded underlines when it makes the index.

Although a module begins with TEX text and ends with Pascal text, we have noted that the dividing line isn't sharp, since Pascal text can be included in TEX text if it is enclosed in '`|...|`'. Conversely, TEX text also appears frequently within Pascal text, because everything in comments (i.e., between left and right braces) is treated as TEX text. Furthermore, a module name consists of TEX text; thus, a WEB file typically involves constructions like '`if x = 0 then @<Empty the |buffer| array@>`' where we go back and forth between Pascal and TEX conventions in a natural way.

Macros. A WEB programmer can define three kinds of macros to make the programs shorter and more readable:

'`@d` *identifier* `=` *constant*' defines a *numeric* macro, allowing TANGLE to do rudimentary arithmetic.

'`@d` *identifier* `==` Pascal text' defines a *simple* macro, where the identifier will be replaced by the Pascal text when TANGLE produces its output.

'`@d` *identifier* `(#) ==` Pascal text' defines a *parametric* macro, where the identifier will be replaced by the Pascal text and where occurrences of `#` in that Pascal text will be replaced by an argument.

In all three cases, the identifier must have length greater than one; it must not be a single letter.

Numeric macros are subject to the following restrictions: (1) The identifier must be making its first appearance in the WEB file; a numeric macro must be defined before it is used. (2) The right-hand side of the numeric definition must be made entirely from integer constants, numeric macros, preprocessed strings (see below), and plus signs or minus signs. No other operations or symbols are allowed, not even parentheses, except that Pascal-like comments (enclosed in braces) can appear. Indeed, comments are recommended, since it is usually wise to give a brief explanation of the significance of each identifier as it is defined. (3) The numeric value must be less than $2^{15} = 32768$ in absolute value. (For larger values, you can use '`==`' in place of '`=`', thus making use of a simple macro instead of a numeric one. Note, however, that simple macros sometimes have a different effect. For example, consider the three definitions '`@d n1=2 @d n2=2+n1 @d n3==2+n1`'; then '`x-n2`' will expand into '`x-4`', while '`x-n3`' will expand into '`x-2+2`' which is quite different! It is wise to include parentheses in non-numeric macros, e.g., '`@d n3==(2+n1)`', to avoid such errors.)

When constants are connected by plus signs or minus signs in a Pascal program, TANGLE does the arithmetic before putting the constant into the output file. Therefore it is permissible to say, for example, '**array** $[0\,..\,size-1]$' if *size* has been declared as a macro; note that Pascal doesn't allow this kind of compile-time arithmetic if *size* is a **constant** quantity in the program. Another use of TANGLE's arithmetic is to make **case** statement labels such as '$flag+1$' legitimate. Of course, it is improper to change `2+2` into `4` without looking at the surrounding context; many counterexamples exist, such as the phrases '`-2+2`', '`x/2+2`', and '`2+2E5`'. The program for TANGLE, in the appendix, gives precise details about this conversion, which TANGLE does only when it is safe.

The right-hand sides of simple and parametric macros are required to have balanced parentheses, and the Pascal texts of modules must have balanced parentheses too. Therefore when the argument to a parametric macro appears in parentheses, both parentheses will belong to the same Pascal text.

The appendices to this report contain hundreds of typical examples of the usefulness of WEB macros, so it is not necessary to dwell on the subject here. However, the reader should know that WEB's apparently

primitive macro capabilities can actually do a lot of rather surprising things. Here is a construction that sheds further light on what is possible: After making the definitions

```
@d two_cases(#)==case j of 1:#(1); 2:#(2); end
@d reset_file(#)==reset(input_file@&#)
```

one can write '`two_cases(reset_file)`' and the resulting Pascal output will be

```
case j of 1:reset(input_file1); 2:reset(input_file2); end
```

(but in uppercase letters and with _'s removed). The '`@&`' operation used here joins together two adjacent tokens into a single token, as explained later; otherwise the Pascal file would contain a space between `input_file` and the digit that followed it. This trick can be used to provide the effect of an array of files, if you are unfortunate enough to have a Pascal compiler that doesn't allow such arrays. Incidentally, the cross-reference index made by `WEAVE` from this example would contain the identifier *input_file* but it would not contain *input_file1* or *input_file2*. Furthermore, `TANGLE` would not catch the error that `INPUTFILE1` and `INPUTFILE2` both begin with the same nine letters; one should be more careful when using '`@&`'! But such aspects of the construction in this trick are peripheral to our main point, which is that a parametric macro name without arguments can be used as an argument to another parametric macro.

Although `WEB`'s macros are allowed to have at most one parameter, the following example shows that this is not as much of a restriction as it may seem at first. Let *amac* and *bmac* be any parametric macros, and suppose that we want to get the effect of

```
@d cmac(#1,#2) == amac(#1) bmac(#2)
```

which `WEB` doesn't permit. The solution is to make the definitions

```
@d cmac(#) == amac(#) dmac
@d dmac(#) == bmac(#)
```

and then to say '`cmac(x)(y)`'.

There is one restriction in the generality of `WEB`'s parametric macros, however: the argument to a parametric macro must not come from the expansion of a macro that has not already been "started." For example, here is one of the things `WEB` cannot handle:

```
@d arg == (p)
@d identity(#) == #
@p identity arg
```

In this case `TANGLE` will complain that the `identity` macro is not followed by an argument in parentheses.

The `WEB` language has another feature that is somewhat similar to a numeric macro. A *preprocessed string* is a string that is like a Pascal string but delimited by double-quote marks (`"`) instead of single-quotes. Double-quote marks inside of such strings are indicated by giving two double-quotes in a row. If a preprocessed string is of length one (e.g., `"A"` or `""""`), it will be treated by `TANGLE` as equivalent to the corresponding ASCII-code integer (e.g., `65` or `34`). And if a preprocessed string is not of length one, it will be converted into an integer equal to 256 or more. A *string pool* containing all such strings will be written out by the `TANGLE` processor; this string pool file consists of string 256, then string 257, etc., where each string is followed by an end-of-line and prefixed by two decimal digits that define its length. Thus, for example, the empty string `""` would be represented in the string pool file by a line containing the two characters '`00`', while the string `"""String"""` would be represented by '`08"String"`'. A given string appears at most once in the string pool; the use of such a pool makes it easier to cope with Pascal's restrictions on string manipulation. The string pool ends with '`*nnnnnnnnn`', where `nnnnnnnnn` is a decimal number called the *string pool check sum*. If any string changes, the check sum almost surely changes too; thus, the '`@$`' feature described below makes it possible for a program to assure itself that it is reading its own string pool.

Here is a simple example that combines numeric macros with preprocessed strings of length one:

```
@d upper_case_Y = "Y"
@d case_difference = -"y"+upper_case_Y
```

The result is to define $upper\_case\_Y = 89$, $case\_difference = -32$.

Control codes. We have seen several magic uses of '@' signs in WEB files, and it is time to make a systematic study of these special features. A WEB *control code* is a two-character combination of which the first is '@'.

Here is a complete list of the legal control codes. The letters L, T, P, M, C, and/or S following each code indicate whether or not that code is allowable in limbo, in TEX text, in Pascal text, in module names, in comments, and/or in strings. A bar over such a letter means that the control code terminates the present part of the WEB file; for example, $\overline{L}$ means that this control code ends the limbo material before the first module.

@@ $[C, L, M, P, S, T]$ A double @ denotes the single character '@'. This is the only control code that is legal in limbo, in comments, and in strings.

@␣ $[\overline{L}, \overline{P}, \overline{T}]$ This denotes the beginning of a new (unstarred) module. A tab mark or end-of-line (carriage return) is equivalent to a space when it follows an @ sign.

@* $[\overline{L}, \overline{P}, \overline{T}]$ This denotes the beginning of a new starred module, i.e., a module 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 WEAVE and TANGLE read a @*, they print an asterisk on the terminal followed by the current module number, so that the user can see some indication of progress. The very first module should be starred.

@d $[\overline{P}, \overline{T}]$ Macro definitions begin with @d (or @D), followed by the Pascal text for one of the three kinds of macros, as explained earlier.

@f $[\overline{P}, \overline{T}]$ Format definitions begin with @f (or @F); they cause WEAVE to treat identifiers in a special way when they appear in Pascal text. The general form of a format definition is '@f l == r', followed by an optional comment enclosed in braces, where l and r are identifiers; WEAVE will subsequently treat identifier l as it currently treats r. This feature allows a WEB programmer to invent new reserved words and/or to unreserve some of Pascal's reserved identifiers. The definition part of each module consists of any number of macro definitions (beginning with @d) and format definitions (beginning with @f), intermixed in any order.

@p $[\overline{P}, \overline{T}]$ The Pascal part of an unnamed module begins with @p (or @P). This causes TANGLE to append the following Pascal code to the initial program text T_0 as explained above. The WEAVE processor does not cause a '@p' to appear explicitly in the TEX output, so if you are creating a WEB file based on a TEX-printed WEB documentation you have to remember to insert @p in the appropriate places of the unnamed modules.

@< $[P, \overline{T}]$ A module name begins with @< followed by TEX text followed by @>; the TEX text should not contain any WEB control codes except @@, unless these control codes appear in Pascal text that is delimited by |...|. The module name may be abbreviated, after its first appearance in a WEB file, by giving any unique prefix followed by ..., where the three dots immediately precede the closing @>. No module name should be a prefix of another. Module names may not appear in Pascal text that is enclosed in |...|, nor may they appear in the definition part of a module (since the appearance of a module name ends the definition part and begins the Pascal part).

@' $[P, T]$ This denotes an octal constant, to be formed from the succeeding digits. For example, if the WEB file contains '@'100', the TANGLE processor will treat this an equivalent to '64'; the constant will be formatted as "*'100*" in the TEX output produced via WEAVE. You should use octal notation only for positive constants; don't try to get, e.g., -1 by saying '@'777777777777'.

@" $[P, T]$ A hexadecimal constant; '@"D0D0' tangles to 53456 and weaves to '"D0D0'.

@$ $[P]$ This denotes the string pool check sum.

@{ $[P]$ The beginning of a "meta comment," i.e., a comment that is supposed to appear in the Pascal code, is indicated by @{ in the WEB file. Such delimiters can be used as isolated symbols in macros or modules, but they should be properly nested in the final Pascal program. The TANGLE processor will convert '@{' into '{' in the Pascal output file, unless the output is already part of a meta-comment; in the latter case '@{' is converted into '[', since Pascal does not allow nested comments. Incidentally, module numbers are automatically inserted as meta-comments into the Pascal program, in order to help correlate the outputs of WEAVE and TANGLE (see Appendix C). Meta-comments can be used to put conditional text

into a Pascal program; this helps to overcome one of the limitations of WEB, since the simple macro processing routines of TANGLE do not include the dynamic evaluation of boolean expressions.

@} [P] The end of a "meta comment" is indicated by '@}'; this is converted either into '}' or ']' in the Pascal output, according to the conventions explained for @{ above.

@& [P] The @& operation causes whatever is on its left to be adjacent to whatever is on its right, in the Pascal output. No spaces or line breaks will separate these two items. However, the thing on the left should not be a semicolon, since a line break might occur after a semicolon.

@^ [P,T] The "control text" that follows, up to the next '@>', will be entered into the index together with the identifiers of the Pascal program; this text will appear in roman type. For example, to put the phrase "system dependencies" into the index, you can type '@^system dependencies@>' in each module that you want to index as system dependent. A control text, like a string, must end on the same line of the WEB file as it began. Furthermore, no WEB control codes are allowed in a control text, not even @@. (If you need an @ sign you can get around this restriction by typing '\AT!'.)

@. [P,T] The "control text" that follows will be entered into the index in typewriter type; see the rules for '@^', which is analogous.

@: [P,T] The "control text" that follows will be entered into the index in a format controlled by the TEX macro '\9', which the user should define as desired; see the rules for '@^', which is analogous.

@t [P] The "control text" that follows, up to the next '@>', will be put into a TEX \hbox and formatted along with the neighboring Pascal program. This text is ignored by TANGLE, but it can be used for various purposes within WEAVE. For example, you can make comments that mix Pascal and classical mathematics, as in '$size < 2^{15}$', by typing '|size < @t2^{15}@>|'. A control text must end on the same line of the WEB file as it began, and it may not contain any WEB control codes.

@= [P] The "control text" that follows, up to the next '@>', will be passed verbatim to the Pascal program.

@\ [P] Force end-of-line here in the Pascal program file.

@! [P,T] The module 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 modules where an identifier is defined, or where it is explained in some special way, from the modules 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 WEAVE just after the reserved words **function**, **procedure**, **program**, and **var**, and just after @d and @f. But you should insert your own '@!' before the definitions of types, constants, variables, parameters, and components of records and enumerated types that are not covered by this implicit convention, if you want to improve the quality of the index that you get.

@? [P,T] This cancels an implicit (or explicit) '@!', so that the next index entry will not be underlined.

@, [P] This control code inserts a thin space in WEAVE's output; it is ignored by TANGLE. Sometimes you need this extra space if you are using macros in an unusual way, e.g., if two identifiers are adjacent.

@/ [P] This control code causes a line break to occur within a Pascal program formatted by WEAVE; it is ignored by TANGLE. 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. Caution: '@/' should be used only after statements or clauses, not in the middle of an expression; use @| in the middle of expressions, in order to keep WEAVE's parser happy.

@| [P] This control code specifies an optional line break in the midst of an expression. For example, if you have a long condition between **if** and **then**, or 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.

@# [P] 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 procedure definitions or between groups of macro definitions that are logically separate but within the same module.

`@+` [*P*] This control code cancels a line break that might otherwise be inserted by `WEAVE`, e.g., before the word '**else**', if you want to put a short if-then-else construction on a single line. It is ignored by `TANGLE`.

`@;` [*P*] This control code is treated like a semicolon, for formatting purposes, except that it is invisible. You can use it, for example, after a module name when the Pascal text represented by that module name ends with a semicolon.

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

Because of the rules by which every module is broken into three parts, the control codes '`@d`', '`@f`', and '`@p`' are not allowed to occur once the Pascal part of a module has begun.

Additional features and caveats.

1. The character pairs '`(*`', '`*)`', '`(.`', and '`.)`' are converted automatically in Pascal text as though they were '`@{`', '`@}`', '`[`', and '`]`', respectively, except of course in strings. Furthermore in certain installations of `WEB` that have an extended character set, the characters '≠', '≤', '≥', '←', '≡', '∧', '∨', '¬', and '∈' can be typed as abbreviations for '`<>`', '`<=`', '`>=`', '`:=`', '`==`', '`and`', '`or`', '`not`', and '`in`', respectively. However, the latter abbreviations are not used in the standard versions of `WEAVE.WEB` and `TANGLE.WEB` that are distributed to people who are installing `WEB` on other computers, and the programs are designed to produce only standard ASCII characters as output if the input consists entirely of ASCII characters.

2. If you have an extended character set, all of the characters listed in Appendix C of *The TEXbook* can be used in strings. 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 `WEAVE` is broken into lines having at most 80 characters each. The algorithm that does this line breaking is unaware of TEX's convention about comments following '`%`' signs on a line. When TEX text is being copied, the existing line breaks are copied as well, so there is no problem with '`%`' signs unless the original `WEB` file contains a line more than eighty characters long or a line with Pascal text in `|...|` that expands to more than eighty characters long. Such lines should not have '`%`' signs.

4. Pascal 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 `WEAVE`. 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 `WEAVE` does, but the general mechanism is somewhat complex because it must handle much more than Pascal 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 adhere to the following restrictions:

a) Comments in Pascal text should appear only after statements or clauses; i.e., after semicolons, after reserved words like **then** and **do**, or before reserved words like **end** and **else**. Otherwise `WEAVE`'s parsing method may well get mixed up.

b) Don't enclose long Pascal texts in `|...|`, since the indentation and line breaking codes are omitted when the `|...|` text is translated from Pascal to TEX. Stick to simple expressions or statements.

5. Comments and module names are not permitted in `|...|` text. After a '`|`' signals the change from TEX text to Pascal text, the next '`|`' that is not part of a string or control text ends the Pascal text.

6. A comment must have properly nested occurrences of left and right braces, otherwise `WEAVE` and `TANGLE` will not know where the comment ends. However, the character pairs '`\{`' and '`\}`' do not count as left and right braces in comments, and the character pair '`\|`' does not count as a delimiter that begins Pascal text. (The actual rule is that a character after '`\`' is ignored; hence in '`\\{`' the left brace *does* count.) At present,

TANGLE and WEAVE treat comments in slightly different ways, and it is necessary to satisfy both conventions: TANGLE ignores '|' characters entirely, while WEAVE uses them to switch between TEX text and Pascal text. Therefore, a comment that includes a brace in a string in |...|—e.g., '{ look at this |"{"| }'—will be handled correctly by WEAVE, but TANGLE will think there is an unmatched left brace. In order to satisfy both processors, one can write '{ look at this \leftbrace\ }', after setting up'\def\leftbrace{|"{"|}'.

7. Reserved words of Pascal must appear entirely in lowercase letters in the WEB file; otherwise their special nature will not be recognized by WEAVE. You could, for example, have a macro named *END* and it would not be confused with Pascal's **end**.

However, you may not want to capitalize macro names just to distinguish them from other identifiers. Here is a way to unreserve Pascal's reserved word '**type**' and to substitute another word '**mtype**' in the WEB file.

```
@d type(#) == mem[#].t
@d mtype == t @& y @& p @& e
@f mtype == type
@f type == true
```

In the output of TANGLE, the macro mtype now produces 'TYPE' and the macro type(x) now produces 'MEM[X].T'. In the output of WEAVE, these same inputs produce **mtype** and *type*(x), respectively.

8. The @f feature allows you to define one identifier to act like another, and these format definitions are carried out sequentially, as the example above indicates. However, a given identifier has only one printed format throughout the entire document (and this format will even be used before the @f that defines it). The reason is that WEAVE operates in two passes; it processes @f's and cross-references on the first pass and it does the output on the second.

9. You may want some @f formatting that doesn't correspond to any existing reserved word. In that case, WEAVE could be extended in a fairly obvious way to include new "reserved words" in its vocabulary. The identifier '**xclause**' has in fact been included already as a reserved word, so that it can be used to format the '**loop**' macro, where '**loop**' is defined to be equivalent to '**while** *true* **do**'.

10. Sometimes it is desirable to insert spacing into Pascal 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 webmac 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.)

11. WEAVE and TANGLE 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 WEB 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 WEB file that has been received from elsewhere (e.g., TANGLE.WEB or WEAVE.WEB 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.

Appendices. The basic ideas of WEB can be understood most easily by looking at examples of "real" programs. Appendix A shows the WEB input that generated modules 55–59 of the WEAVE program; Appendix B shows the corresponding TEX code output by WEAVE; and Appendix C shows excerpts from the corresponding Pascal code output by TANGLE.

The complete webs for WEAVE and TANGLE appear as the bulk of this report, in Appendices D and E. The reader should first compare Appendix A to the corresponding portion of Appendix D; then the same material should be compared to Appendices B and C. Finally, if time permits, the reader may enjoy studying the complete programs in Appendices D and E, since WEAVE and TANGLE contain several interesting aspects, and since an attempt has been made in these appendices to evolve a style of programming that makes good use of the WEB language.

Finally, Appendix F is the 'webmac' file that sets TEX up to accept the output of WEAVE; Appendix G discusses how to use some of its macros to vary the output formats; and Appendix H discusses what needs to be done when WEAVE and TANGLE are installed in a new operating environment.

Performance statistics. The programs in Appendices D and E will optionally keep statistics on how much memory they require. Here is what they once printed out when processing themselves:

```
TANGLE applied to TANGLE (cpu time 15 sec)
  Memory usage statistics:
  456 names, 215 replacement texts;
  3396+3361 bytes, 6685+7329+5805 tokens.
TANGLE applied to WEAVE (cpu time 30 sec)
  Memory usage statistics:
  692 names, 339 replacement texts;
  4576+4294 bytes, 10184+9875+9150 tokens.
WEAVE applied to TANGLE (cpu time 45 sec)
  Memory usage statistics:  478 names, 2045 cross references, 4159+3729 bytes;
  parsing required 684 scraps, 1300 texts, 3766 tokens, 119 levels;
  sorting required 34 levels.
WEAVE applied to WEAVE (cpu time 65 sec)
  Memory usage statistics:  737 names, 3306 cross references, 4896+4962 bytes;
  parsing required 684 scraps, 1300 texts, 3766 tokens, 119 levels;
  sorting required 73 levels.
```

The cpu time for Pascal to process TANGLE.PAS was approximately 13 seconds, and WEAVE.PAS took approximately 26 seconds; thus the tangling time was slightly more than the compiling time. The cpu time for TEX to process TANGLE.TEX was approximately 500 seconds, and WEAVE.TEX took approximately 750 seconds (i.e., about 7 seconds per printed page, where these pages are substantially larger than the pages in a normal book). All cpu times quoted are for a DECsystem-10.

The file TANGLE.WEB is about 125K characters long; TANGLE reduces it to a file TANGLE.PAS whose size is about 42K characters, while WEAVE expands it to a file TANGLE.TEX of about 185K. The corresponding file sizes for WEAVE.WEB, WEAVE.PAS, and WEAVE.TEX are 180K, 89K, and 265K.

The much larger file TEX.WEB led to the following numbers:

```
TANGLE applied to TEX (cpu time 110 sec)
  Memory usage statistics:
  3750 names, 1768 replacement texts;
  41895+41053 bytes, 42378+45074+41091 tokens.
WEAVE applied to TEX (cpu time 270 sec)
  Memory usage statistics:  3412 names, 19699 cross references, 37900+40232 bytes;
  parsing required 685 scraps, 1303 texts, 3784 tokens, 104 levels;
  sorting required 52 levels.
```

Pascal did TEX.PAS in about 75 seconds; TEX did TEX.TEX in about 3600.

O, what a tangled web we weave
When first we practise to deceive!
—SIR WALTER SCOTT, *Marmion* 6:17 (1808)

O, what a tangled WEB we weave
When TEX we practise to conceive!
—RICHARD PALAIS (1982)

Appendix A. This excerpt from WEAVE.WEB produced modules 55–59 in Appendix D. Note that some of the lines are indented to show the program structure. The indentation is ignored by WEAVE and TANGLE, but users find that WEB files are quite readable if they have some such indentation.

```
@* Searching for identifiers.
The hash table described above is updated by the |id_lookup| procedure,
which finds a given identifier and returns a pointer to its index in
|byte_start|. The identifier is supposed to match character by character
and it is also supposed to have a given |ilk| code; the same name may be
present more than once if it is supposed to appear in the index with
different typesetting conventions.
If the identifier was not already present, it is inserted into the table.

Because of the way \.{WEAVE}'s scanning mechanism works, it is most convenient
to let |id_lookup| search for an identifier that is present in the |buffer|
array. Two other global variables specify its position in the buffer: the
first character is |buffer[id_first]|, and the last is |buffer[id_loc-1]|.

@<Glob...@>=
@!id_first:0..long_buf_size; {where the current identifier begins in the buffer}
@!id_loc:0..long_buf_size; {just after the current identifier in the buffer}
@#
@!hash:array [0..hash_size] of sixteen_bits; {heads of hash lists}

@ Initially all the hash lists are empty.

@<Local variables for init...@>=
@!h:0..hash_size; {index into hash-head array}

@ @<Set init...@>=
for h:=0 to hash_size-1 do hash[h]:=0;

@ Here now is the main procedure for finding identifiers (and index
entries).  The parameter |t| is set to the desired |ilk| code. The
identifier must either have |ilk=t|, or we must have
|t=normal| and the identifier must be a reserved word.

@p function id_lookup(@!t:eight_bits):name_pointer; {finds current identifier}
label found;
var i:0..long_buf_size; {index into |buffer|}
@!h:0..hash_size; {hash code}
@!k:0..max_bytes; {index into |byte_mem|}
@!w:0..ww-1; {row of |byte_mem|}
@!l:0..long_buf_size; {length of the given identifier}
@!p:name_pointer; {where the identifier is being sought}
begin l:=id_loc-id_first; {compute the length}
@<Compute the hash code |h|@>;
@<Compute the name location |p|@>;
if p=name_ptr then @<Enter a new name into the table at position |p|@>;
id_lookup:=p;
end;

@ A simple hash code is used: If the sequence of
ASCII codes is $c_1c_2\ldots c_m$, its hash value will be
$$(2^{n-1}c_1+2^{n-2}c_2+\cdots+c_n)\,\bmod\,|hash_size|.$$

@<Compute the hash...@>=
h:=buffer[id_first]; i:=id_first+1;
while i<id_loc do
  begin h:=(h+h+buffer[i]) mod hash_size; incr(i);
  end
```

Appendix B. This excerpt from WEAVE.TEX corresponds to Appendix A.

```
\N55.  Searching for identifiers.
The hash table described above is updated by the \\{id\_lookup} procedure,
which finds a given identifier and returns a pointer to its index in
\\{byte\_start}. The identifier is supposed to match character by character
and it is also supposed to have a given \\{ilk} code; the same name may be
present more than once if it is supposed to appear in the index with
different typesetting conventions.
If the identifier was not already present, it is inserted into the table.

Because of the way \.{WEAVE}'s scanning mechanism works, it is most convenient
to let \\{id\_lookup} search for an identifier that is present in the %
\\{buffer}
array. Two other global variables specify its position in the buffer: the
first character is $\\{buffer}[\\{id\_first}]$, and the last is $\\{buffer}[%
\\{id\_loc}-1]$.

\Y\P$\4\X9:Globals in the outer block\X\mathrel{+}\S$\6
\4\\{id\_first}: \37$0\to\\{long\_buf\_size}$;\C{where the current identifier
begins in the buffer}\6
\4\\{id\_loc}: \37$0\to\\{long\_buf\_size}$;\C{just after the current
identifier in the buffer}\7
\4\\{hash}: \37\&{array} $[0\to\\{hash\_size}]$ \1\&{of}\5
\\{sixteen\_bits};\C{heads of hash lists}\2\par
\fi

\M56. Initially all the hash lists are empty.

\Y\P$\4\X16:Local variables for initialization\X\mathrel{+}\S$\6
\4\|h: \37$0\to\\{hash\_size}$;\C{index into hash-head array}\par
\fi

\M57. \P$\X10:Set initial values\X\mathrel{+}\S$\6
\&{for} $\|h\K0\mathrel{\&{to}}\\{hash\_size}-1$ \1\&{do}\5
$\\{hash}[\|h]\K0$;\2\par
\fi

\M58. Here now is the main procedure for finding identifiers (and index
entries).  The parameter \|t is set to the desired \\{ilk} code. The
identifier must either have $\\{ilk}=\|t$, or we must have
$\|t=\\{normal}$ and the identifier must be a reserved word.

\Y\P\4\&{function}\1\  \37$\\{id\_lookup}(\|t:\\{eight\_bits})$: \37\\{name%
\_pointer};\C{finds current identifier}\6
\4\&{label} \37\\{found};\6
\4\&{var} \37\|i: \37$0\to\\{long\_buf\_size}$;\C{index into \\{buffer}}\6
\|h: \37$0\to\\{hash\_size}$;\C{hash code}\6
\|k: \37$0\to\\{max\_bytes}$;\C{index into \\{byte\_mem}}\6
\|w: \37$0\to\\{ww}-1$;\C{row of \\{byte\_mem}}\6
\|l: \37$0\to\\{long\_buf\_size}$;\C{length of the given identifier}\6
\|p: \37\\{name\_pointer};\C{where the identifier is being sought}\2\6
\&{begin} \37$\|l\K\\{id\_loc}-\\{id\_first}$;\C{compute the length}\6
\X59:Compute the hash code \|h\X;\6
\X60:Compute the name location \|p\X;\6
\&{if} $\|p=\\{name\_ptr}$ \1\&{then}\5
\X62:Enter a new name into the table at position \|p\X;\2\6
$\\{id\_lookup}\K\|p$;\6
\&{end};\par
\fi

\M59. A simple hash code is used: If the sequence of
ASCII codes is $c_1c_2\ldots c_m$, its hash value will be
$$(2^{n-1}c_1+2^{n-2}c_2+\cdots+c_n)\,\bmod\,\\{hash\_size}.$$

\Y\P$\4\X59:Compute the hash code \|h\X\S$\6
$\|h\K\\{buffer}[\\{id\_first}]$;\5
$\|i\K\\{id\_first}+1$;\6
\&{while} $\|i<\\{id\_loc}$ \1\&{do}\6
\&{begin} \37$\|h\K(\|h+\|h+\\{buffer}[\|i])\mathbin{\&{mod}}\\{hash\_size}$;\5
$\\{incr}(\|i)$;\6
\&{end}\2\par
\U section~58.\fi
```

Appendix C. The TANGLE processor converts WEAVE.WEB into a syntactically correct (but not very pretty) Pascal program WEAVE.PAS. The first three and last two lines of output are shown here, together with the lines of code generated by modules 55–62 and the environments of those lines. There are 1559 lines in all; the notation '. . .' stands for portions that are not shown.

Note that, for example, the code corresponding to module 55 begins with '{55:}' and ends with '{:55}'; the code from modules 59–62 has been tangled into the code from module 58.

```
{2:}{4:}{$C-,A+,D-}{[$C+,D+]}{:4}
PROGRAM WEAVE(WEBFILE,CHANGEFILE,TEXFILE);LABEL 9999;CONST{8:}
MAXBYTES=45000;MAXNAMES=5000;MAXMODULES=2000;HASHSIZE=353;BUFSIZE=100;
     . . .
TOKPTR:0..MAXTOKS;{MAXTOKPTR,MAXTXTPTR:0..MAXTOKS;}{:53}{55:}
IDFIRST:0..LONGBUFSIZE;IDLOC:0..LONGBUFSIZE;
HASH:ARRAY[0..HASHSIZE]OF SIXTEENBITS;{:55}{63:}CURNAME:NAMEPOINTER;
     . . .
PROCEDURE INITIALIZE;VAR{16:}I:0..127;{:16}{40:}WI:0..1;{:40}{56:}
H:0..HASHSIZE;{:56}{247:}C:ASCIICODE;{:247}BEGIN{10:}HISTORY:=0;{:10}
     . . .
TOKPTR:=1;TEXTPTR:=1;TOKSTART[0]:=1;TOKSTART[1]:=1;{MAXTOKPTR:=1;
MAXTXTPTR:=1;}{:54}{57:}FOR H:=0 TO HASHSIZE-1 DO HASH[H]:=0;{:57}{94:}
SCANNINGHEX:=FALSE;{:94}{102:}MODTEXT[0]:=32;{:102}{124:}OUTPTR:=1;
     . . .
IF R=0 THEN XREF[P]:=XREFPTR ELSE XMEM[R].XLINKFIELD:=XREFPTR;END;{:51}
{58:}FUNCTION IDLOOKUP(T:EIGHTBITS):NAMEPOINTER;LABEL 31;
VAR I:0..LONGBUFSIZE;H:0..HASHSIZE;K:0..MAXBYTES;W:0..1;
L:0..LONGBUFSIZE;P:NAMEPOINTER;BEGIN L:=IDLOC-IDFIRST;{59:}
H:=BUFFER[IDFIRST];I:=IDFIRST+1;
WHILE I<IDLOC DO BEGIN H:=(H+H+BUFFER[I])MOD HASHSIZE;I:=I+1;END{:59};
{60:}P:=HASH[H];
WHILE P<>0 DO BEGIN IF(BYTESTART[P+2]-BYTESTART[P]=L)AND((ILK[P]=T)OR((T
=0)AND(ILK[P]>3)))THEN{61:}BEGIN I:=IDFIRST;K:=BYTESTART[P];W:=P MOD 2;
WHILE(I<IDLOC)AND(BUFFER[I]=BYTEMEM[W,K])DO BEGIN I:=I+1;K:=K+1;END;
IF I=IDLOC THEN GOTO 31;END{:61};P:=LINK[P];END;P:=NAMEPTR;
LINK[P]:=HASH[H];HASH[H]:=P;31:{:60};IF P=NAMEPTR THEN{62:}
BEGIN W:=NAMEPTR MOD 2;
IF BYTEPTR[W]+L>MAXBYTES THEN BEGIN WRITELN(TERMOUT);
WRITE(TERMOUT,'! Sorry, ','byte memory',' capacity exceeded');ERROR;
HISTORY:=3;JUMPOUT;END;
IF NAMEPTR+2>MAXNAMES THEN BEGIN WRITELN(TERMOUT);
WRITE(TERMOUT,'! Sorry, ','name',' capacity exceeded');ERROR;HISTORY:=3;
JUMPOUT;END;I:=IDFIRST;K:=BYTEPTR[W];
WHILE I<IDLOC DO BEGIN BYTEMEM[W,K]:=BUFFER[I];K:=K+1;I:=I+1;END;
BYTEPTR[W]:=K;BYTESTART[NAMEPTR+2]:=K;NAMEPTR:=NAMEPTR+1;ILK[P]:=T;
XREF[P]:=0;END{:62};IDLOOKUP:=P;END;{:58}{66:}
FUNCTION MODLOOKUP(L:SIXTEENBITS):NAMEPOINTER;LABEL 31;VAR C:0..4;
     . . .
WRITE(TERMOUT,'(That was a fatal error, my friend.)');END;END{:263};
END.{:261}
```

Appendix F: The `webmac.tex` file. This is the file that extends "plain TEX" format in order to support the features needed by the output of `WEAVE`.

```
% standard macros for WEB listings (in addition to PLAIN.TEX)
\xdef\fmtversion{\fmtversion+WEBMAC4.0} % identifies current set of macros
\parskip 0pt % no stretch between paragraphs
\parindent 1em % for paragraphs and for the first line of Pascal text

\font\eightrm=cmr8 \let\sc=\eightrm % NOT a caps-and-small-caps font!
\let\mainfont=\tenrm
\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)

\def\\#1{\hbox{\it#1\/\kern.05em}} % italic type for identifiers
\def\|#1{\hbox{$#1$}} % one-letter identifiers look a bit better this way
\def\&#1{\hbox{\bf#1\/}} % boldface type for reserved words
\def\.#1{\hbox{\tentex % typewriter type for strings
  \let\\=\BS % backslash in a string
  \let\'=\RQ % right quote in a string
  \let\`=\LQ % left quote 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
  #1}}
\def\#{\hbox{\tt\char`\#}} % parameter sign
\def\${\hbox{\tt\char`\$}} % dollar sign
\def\%{\hbox{\tt\char`\%}} % percent sign
\def\^{\ifmmode\mathchar"222 \else\char`^ \fi} % pointer or hat
% circumflex accents can be obtained from \^^D instead of \^
\def\AT!{@} % at sign for control text

\chardef\AM=`\& % ampersand character in a string
\chardef\BS=`\\ % backslash in a string
\chardef\LB=`\{ % left brace in a string
\def\LQ{{\tt\char'22}} % left quote in a string
\chardef\RB=`\} % right brace in a string
\def\RQ{{\tt\char'23}} % right quote 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

\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

\let\yskip=\smallskip
\def\to{\mathrel{.\,.}} % double dot, used only in math mode
\def\note#1#2.{\Y\noindent{\hangindent2em\baselineskip10pt\eightrm#1~#2.\par}}
\def\lapstar{\rlap{*}}
\def\startsection{\Q\noindent{\let\*=\lapstar\bf\modstar.\quad}}
\def\defin#1{\global\advance\ind by 2 \1\&{#1 }} % begin ‘define’ or ‘format’
\def\A{\note{See also section}} % crossref for doubly defined section name
\def\As{\note{See also sections}} % crossref for multiply defined section name
\def\B{\mathopen{\.{@\{}}} % begin controlled comment
\def\C#1{\ifmmode\gdef\XX{\null$\null}\else\gdef\XX{}\fi % Pascal comments
  \XX\hfil\penalty-1\hfilneg\quad$\{\,$#1$\,\}$\XX}
\def\D{\defin{define}} % macro definition
\def\E{\cdot10^} % exponent in floating point constant
\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
\def\H#1{\hbox{\rm\char"7D\tt#1}} % hexadecimal constant
\let\I=\ne % unequal sign
\def\J{\.{@\&}} % TANGLE’s join operation
\let\K=\gets % left arrow
\let\L=\le % less than or equal sign
\outer\def\M#1.{\MN#1.\ifon\vfil\penalty-100\vfilneg % beginning of section
  \vskip12ptminus3pt\startsection\ignorespaces}
\outer\def\N#1.#2.{\MN#1.\vfil\eject % beginning of starred section
  \def\rhead{\uppercase{\ignorespaces#2}} % define running headline
  \message{*\modno} % progress report
  \edef\next{\write\cont{\Z{#2}{\modno}{\the\pageno}}}\next % to contents file
  \ifon\startsection{\bf\ignorespaces#2.\quad}\ignorespaces}
\def\MN#1.{\par % common code for \M, \N
  {\xdef\modstar{#1}\let\*=\empty\xdef\modno{#1}}
  \ifx\modno\modstar \onmaybe \else\ontrue \fi \mark{\modno}}
\def\O#1{\hbox{\rm\char’23\kern-.2em\it#1\/\kern.05em}} % octal constant
\def\P{\rightskip=0pt plus 100pt minus 10pt % go into Pascal mode
  \sfcode‘;=3000
  \pretolerance 10000
  \hyphenpenalty 10000 \exhyphenpenalty 10000
  \global\ind=2 \1\ \unskip}
\def\Q{\rightskip=0pt % get out of Pascal mode
  \sfcode‘;=1500 \pretolerance 200 \hyphenpenalty 50 \exhyphenpenalty 50 }
\let\R=\lnot % logical not
\let\S=\equiv % equivalence sign
\def\T{\mathclose{\.{@\}}}} % terminate controlled comment
\def\U{\note{This code is used in section}} % crossref for use of a section
\def\Us{\note{This code is used in sections}} % crossref 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=\let % now you can \send the control sequence \Z
\def\){\hbox{\.{@\$}}} % sign for string pool check sum
\def\]{\hbox{\.{@\\}}} % sign for forced line break
\def\=#1{\kern2pt\hbox{\vrule\vtop{\vbox{\hrule
        \hbox{\strut\kern2pt\.{#1}\kern2pt}}
      \hrule}\vrule}\kern2pt} % verbatim string
\let\~=\ignorespaces
\let\*=*

\def\onmaybe{\let\ifon=\maybe} \let\maybe=\iftrue
\newif\ifon \newif\iftitle \newif\ifpagesaved
\def\lheader{\mainfont\the\pageno\eightrm\qquad\rhead\hfill\title\qquad
  \tensy x\mainfont\topmark} % top line on left-hand pages
\def\rheader{\tensy x\mainfont\topmark\eightrm\qquad\title\hfill\rhead
  \qquad\mainfont\the\pageno} % top line on right-hand pages
\def\page{\box255 }
\def\normaloutput#1#2#3{\ifodd\pageno\hoffset=\pageshift\fi
  \shipout\vbox{
    \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}

\def\rhead{\.{WEB} OUTPUT} % this running head is reset by starred sections
\def\title{} % an optional title can be set by the user
\def\topofcontents{\centerline{\titlefont\title}
  \vfill} % this material will start the table of contents page
\def\botofcontents{\vfill} % this material will end the table of contents page
\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{CONTENTS} % file that gets table of contents info
\def\readcontents{\input CONTENTS}

\newwrite\cont
\output{\setbox0=\page % the first page is garbage
  \openout\cont=\contentsfile
  \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
  \write\cont{} % ensure that the contents file isn't empty
  \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\:##1, {\par\hangindent2em\noindent##1:\kern1em} % index entry
  \let\ttentry=\. \def\.##1{\ttentry{##1\kern.2em}} % give \tt a little room
  \def\[##1]{$\underline{##1}$} % underlined index item
  \rm \rightskip0pt plus 2.5em \tolerance 10000 \let\*=\lapstar
  \hyphenpenalty 10000 \parindent0pt}
\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\rhead{NAMES OF THE SECTIONS}
  \message{Section names:}
  \output{\normaloutput\page\lheader\rheader}
  \setpage
  \def\note##1##2.{\hfil\penalty-1\hfilneg\quad{\eightrm##1 ##2.}}
  \linepenalty=10 % try to conserve lines
  \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\:{\par\hangindent 2em}\let\*=*\let\.=\ttentry}
\def\con{\par\vfill\eject % finish the section names
  \rightskip 0pt \hyphenpenalty 50 \tolerance 200
  \setpage
  \output{\normaloutput\page\lheader\rheader}
  \titletrue % prepare to output the table of contents
  \pageno=\contentspagenumber \def\rhead{TABLE OF CONTENTS}
  \message{Table of contents:}
  \topofcontents
  \line{\hfil Section\hbox to3em{\hss Page}}
  \def\Z##1##2##3{\line{\ignorespaces##1
    \leaders\hbox to .5em{.\hfil}\hfil\ ##2\hbox to3em{\hss##3}}}
  \readcontents\relax % read the contents info
  \botofcontents \end} % print the contents page(s) and terminate
```

Appendix G: How to use WEB macros. The macros in webmac make it possible to produce a variety of formats without editing the output of WEAVE, and the purpose of this appendix is to explain some of the possibilities.

1. Three fonts have been declared in addition to the standard fonts of PLAIN format: 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.

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 WEAVE's translation from Pascal mode.

3. To get typewriter-like type, as when referring to 'WEB', you can use the '\.' macro (e.g., '\.{WEB}'). In the argument to this macro you should insert an additional backslash before the symbols listed as 'special string characters' in the index to WEAVE, 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 '{␣}'.

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

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

were used to prepare the present report; \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. For example, Appendix D was produced after saying '\def\title{WEAVE}'.

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'. (Appendix D was generated that way.)

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. For example, Appendix D was produced with the following definitions:

```
\def\topofcontents{\null\vfill
  \titlefalse % include headline on the contents page
  \def\rheader{\mainfont Appendix␣D\hfil 15}
  \centerline{\titlefont The {\ttitlefont WEAVE} processor}
  \vskip 15pt \centerline{(Version 4)} \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 module. For example, when Appendix D was being generated, a file CONTENTS.TEX containing

```
\Z { Introduction}{1}{16}
\Z { The character set}{11}{19}
```

and similar lines was created. The \topofcontents macro could redefine \Z so that the information appears in another format.

10. Sometimes it is necessary or desirable to divide the output of WEAVE 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 module. 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 webmac 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 CONTENTS};
```

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'. WEAVE 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 WEB file; WEAVE 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 module 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 \modno is set to the number of the module being typeset.

13. If you want to list only the modules that have changed, together with the index, put the command '\let\maybe=\iffalse' in the limbo section before the first module of your WEB 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, \As, \ET, \ETs, \U, \Us, \ch, \fin, and \con. WEAVE itself need not be changed.

Appendix H: Installing the WEB system. Suppose you want to use the WEB programs on your computer, and suppose that you can't simply borrow them from somebody else who has the same kind of machine. Here's what to do:

(1) Get a tape that contains the files WEAVE.WEB, TANGLE.WEB, TANGLE.PAS, and WEBMAC.TEX. The tape will probably also contain an example change file TANGLE.CH.
(2) Look at the sections of TANGLE that are listed under "system dependencies" in the index of Appendix E above, and figure out what changes (if any) will be needed for your system.
(3) Make a change file TANGLE.CH that contains the changes of (2); do not change your copy of TANGLE.WEB, leave it intact. (The rules for change files are explained at the end of the manual just before the appendices; you may want to look at the example change file that arrived with your copy of TANGLE.WEB. It's also a good idea to define all the "switches" like **debug** and **gubed** to be null in your first change files; then you can sure that your compiler will handle all of the code.)
(4) Make the changes of (2) in your copy of TANGLE.PAS. (If these changes are extensive, you might be better off finding some computer that that already has TANGLE running, and making the new TANGLE.PAS from TANGLE.WEB and your TANGLE.CH.)
(5) Use your Pascal compiler to convert your copy of TANGLE.PAS to a running program TANGLE.
(6) Check your changes as follows: Run TANGLE on TANGLE.WEB and your TANGLE.CH, yielding TANGLE.PAS′; make a running program TANGLE′ by applying Pascal to TANGLE.PAS′; run TANGLE′ on TANGLE.WEB and your TANGLE.CH, yielding TANGLE.PAS″; and check that TANGLE.PAS″ is identical to TANGLE.PAS′. Once this test has been passed, you have got a working TANGLE program.
(7) Make a change file WEAVE.CH analogous to (3), but this time consider the system-dependent parts of WEAVE that are listed in the index to Appendix D.
(8) Run TANGLE on WEAVE.WEB and your WEAVE.CH, obtaining WEAVE.PAS.
(9) Use Pascal on WEAVE.PAS to make a running WEAVE program.
(10) Run WEAVE on TANGLE.WEB and TANGLE.CH to produce TANGLE.TEX.
(11) Run TEX on TANGLE.TEX, obtaining a listing analogous to Appendix E. This listing will incorporate your changes.
(12) Run WEAVE on WEAVE.WEB and your WEAVE.CH to produce WEAVE.TEX.
(13) Run TEX on WEAVE.TEX, obtaining a listing analogous to Appendix D that incorporates your changes.

This description assumes that you already have a working TEX82 system. But what if you don't have TEX82? Then you start with a tape that also contains TEX.WEB and plain.tex, and you refer to a hardcopy listing of the TEX82 program corresponding to TEX.WEB. Between steps (10) and (11) you do the following:

(10.1) Make a change file TEX.CH to fix the system dependent portions of TEX.WEB, in a manner analogous to step (2). Since TEX is a much more complex program than WEAVE or TANGLE, there are more system-dependent features to think about, but by now you will be good at making such modifications. Do not make any changes to TEX.WEB.
(10.2) Make an almost-copy of your TEX.CH called INITEX.CH; this one will have the '**init**' and '**tini**' macros redefined in order to make the initialization version of TEX. It also might have smaller font memory and dynamic memory areas, since INITEX doesn't need as much memory for such things; by setting the memory smaller in INITEX, you guarantee that the production system will have a "cushion."
(10.3) Run TANGLE on TEX.WEB and INITEX.CH, obtaining INITEX.PAS and TEX.POOL.
(10.4) Run Pascal on INITEX.PAS, obtaining INITEX.
(10.5) Run INITEX on TEX.POOL, during which run you type '`plain`' and '`\dump`'. This will produce a file plain.fmt containing the data needed to initialize TEX's memory.
(10.6) Run TANGLE on TEX.WEB and the TEX.CH of (10.1), obtaining TEX.PAS.
(10.7) Run Pascal on TEX.PAS, obtaining VIRTEX.
(10.8) If your operating system supports programs whose core images have been saved, run VIRTEX, type '`&plain`', then save the core image and call it TEX. Otherwise, VIRTEX will be your TEX, and it will read '`plain.fmt`' (or some other fmt file) each time you run.

This 21-step process may seem long, but it is actually an oversimplification, since you also need fonts and a way to print the device-independent files that TEX spews out. On the other hand, the total number of steps is not quite so large when you consider that TANGLE-followed-by-Pascal and WEAVE-followed-by-TEX may be regarded as single operations.

If you have only the present report, not a tape, you will have to prepare files WEAVE.WEB and TANGLE.WEB by hand, typing them into the computer by following Appendices D and E. Then you have to simulate the behavior of TANGLE by converting TANGLE.WEB manually into TANGLE.PAS; with a good text editor this takes about six hours. Then you have to correct errors that were made in all this hand work; but still the whole project is not impossibly difficult, because in fact the entire development of WEAVE and TANGLE (including the writing of the programs and this manual) took less than two months of work.