line of the WEB file, the higher priority changefile is used. Priority refers to position within the list of changefiles (f_1 would have a higher priority than f_2).

Conflicts when merging changefiles are inevitable. While significant conflicts are not very likely, since the changes being merged are normally for different purposes and modify different portions of the code, conflicts of a trivial nature occur often. For instance, many WEB programs follow the example of Stanford and output a "banner line" to the terminal to identify the program and its version level, as in:

```
@d banner=='This is WEAVE,
   Version X.X'
```

Nearly all changefiles modify this line to reflect what change they are making to the program, such as:

```
@d banner=='This is WEAVE
   with hyperspace option, ...'
@d banner=='This is MWEAVE,
   Modula-2 WEAVE, ...'
```

for modifications to the logic of the program itself or

```
@d banner=='This is WEAVE,
   VAX/VMS Version ...'
@d banner=='This is WEAVE,
   Microsoft Pascal Version ...'
```

for the various implementation changefiles. However, when multiple changefiles are being merged, the banner line of none of them is correct, since the version of the program actually executing is a combination of the two:

```
@d banner=='This is MWEAVE,
   VAX/VMS Version ...'
```

The \title command in the "limbo" portion of a WEB program falls in the same category as the banner line, since it is also a target common to many changefiles.

The solution to this problem is to create a *third* changefile containing nothing but conflict resolutions. Its change sections would consist only of the composite banner line and title. It should be placed first in the list, so that its changes will override all of the others. Since the conflicts it addresses are expected, the warning messages can be ignored. (It goes without saying that any *unexpected* conflicts which surface must be analyzed to insure that they don't change the logic of the program to an uncompilable or unexecutable state.)

If the sequential approach of TIE is truly needed, the case where one changefile needs to be fully applied before the second one is applied to the result of the first, this can be accomplished serially by using WEBMERGE to create an intermediate WEB file and then applying the second changefile to it. Of course, this does require additional steps, but that's what batch files and command procedures are for.

Hopefully, WEBMERGE should be available from Stanford on the regular distribution tape by the time this reaches print. The WEB files and the VAX implementation files should be available from Stanford and additionally from Kellerman and Smith. For the people who have absolutely no way of reading a magnetic tape, the IBM PC version is available from me on PC floppies for a handling fee. Additionally, the original TANGLE and WEAVE, the MWEB system described elsewhere in this issue, and several of the TEX and METAFONT utility programs (sometimes referred to as TEXware and METAFONTware) are also available on floppy. All of these have change files targeted for Microsoft Pascal running under MS-DOS on the IBM PC, which is my development system. As far as other target computers are concerned, WEBMERGE was cannibalized from TANGLE, so it should be possible to adapt the current implementation-specific changefile for TANGLE without too much difficulty. If you have TANGLE running, you should have no trouble with WEBMERGE.

How to MANGLE Your Software: The WEB System for Modula-2

E. W. (Wayne) Sewell
Software Engineering Specialist

Standard Pascal is an incomplete language from a real-world production software point of view. This is not surprising, since the language was originally designed by Niklaus Wirth as a tool for teaching structured programming, and was never intended for development of production code. The only reason for the widespread use of Pascal is that the various implementors extended the language tremendously when they developed their compilers. VAX Pascal is a good example of a full-featured production compiler. Its many extensions to Pascal allow sophisticated systems to be developed with it. Virtually *every* implementation of Pascal has to extend it in some way, since standard Pascal (as described in Jensen & Wirth) is absolutely

unusable, and ISO Pascal is not much better. While the extensions make Pascal a viable language, portability suffers because each of the implementors extended the language a different way, resulting in a Babel of dialects that is surpassed only by the BASIC language. Porting a program from one Pascal to another is a major effort, even on the same machine. Typical of the problems encountered is the **case** statement. The action to be performed if none of the cases match is not defined in standard Pascal. Since this is a major hole in the language, most implementations try to fill it. Some provide an **else** or **otherwise** clause, others use labels (such as *others:* or *otherwise:*). Whatever mechanism a compiler uses, it is different from what every other compiler uses.

The WEB system tries to counteract the portability problem by using macros for constructs that should have been addressed in the language and then redefining the macros in the implementation-specific change files to generate the correct code, allowing the generic WEB file to remain constant for all implementations. While this makes it possible to write portable Pascal programs, it would still be much less work if the language itself were more standardized.

While WEB does a tremendous job of overcoming the deficiencies of Pascal, there are limits to what can be accomplished. For instance, Pascal does not support separate compilation. A Pascal program is a monolithic block which must be compiled as a unit. *Include files*, which allow a program to be broken up into more than one source file, do not change this fact because the program is still logically one large block and must be compiled as such. Variables not local to a procedure are global to the entire program and are therefore available for accidental modification. Unrelated parts of the program can interact in unexpected ways, especially if the same variable names are used in more than one place. For example, forgetting to declare a variable which should be defined local to a procedure will be detected immediately by the compiler *unless* a variable of a compatible type with the same name is declared globally. The result is that the wrong variable, one unrelated to the procedure, will be modified. Errors of this nature can be very difficult to find. The WEB system can help detect this type of error (if the programmer happens to notice the inconsistencies in the cross-reference listing), but will not prevent it from happening.

The language Modula-2 was designed by Wirth to be the successor to Pascal. Unlike the original Pascal, it was designed to be used for developing real software. Most of the problems with Pascal are corrected by Modula-2, including the **case** problem mentioned above. The syntax is more straightforward, with less likelihood of ambiguities. The most important contribution of Modula-2 is that embodied in the name — the module concept. Modula-2 makes it possible to break up a large programming project into smaller independent pieces, called *modules*, each logically isolated from the others via the software engineering principle of *information hiding*.

The Modula-2 language is much more standardized than Pascal. Since the language is so much more powerful, there is less need to extend it. Input and output, the bane of portability, are completely removed from the language definition itself and are instead banished to library procedures that are more-or-less standardized.

While Modula-2 fixes most of the problems of Pascal and nearly all of the differences between Modula-2 and Pascal are improvements, a couple of the features of the language are steps backward, in my opinion. Case-sensitivity is one of the non-enhancements. In a Modula-2 program, *junk*, *Junk*, and *JUNK* would be considered three different variables. The reason for this change from Pascal, if any, is not obvious. I have never heard a *reasonable* explanation for it. Equally annoying, all of the Modula-2 reserved words are required to be in uppercase. This one almost makes sense, since having the reserved words stand out in this way would make a regular ASCII listing more readable. However, I don't feel that this slight benefit is worth the extra effort involved in writing a program. Using a powerful editor with macro and/or template capability which can fill in the reserved words on behalf of the programmer would make this less painful, but not necessarily enjoyable. I don't wish to give the impression that I am down on Modula-2 because of these issues. It is still my language of choice because the tremendous advantages it provides greatly outweigh the irritations.

MWEB is a version of the WEB system which has been customized for the language Modula-2. Many of the deficiencies of Pascal that are repaired by the WEB system are unnecessary in MWEB, since Modula-2 fixes most of them in the language definition itself. Some examples are the **else** clause on a **case** statement, the standard procedure to increment a variable (INC), and the **loop**, **exit**, and **return** instructions. To counteract the new problems introduced by the language, I designed MWEB to fix Modula-2 in the same way that Modula-2 and standard WEB fix Pascal. The effort expended by

MWEB in this effort is small compared to the lengths necessary to bring Pascal to a usable state. The result of the merger of Modula-2 and MWEB is a programming system that has the advantages of both and few of the disadvantages.

The transformation from WEB to MWEB was comparatively easy — Pascal and Modula-2 are so much alike to begin with, at least syntactically. In fact, Modula-2 is actually less complicated than Pascal and has a cleaner syntax with fewer ambiguities.

MANGLE and MWEAVE were created by modifying their regular WEB counterparts with a standard change file. I wanted to minimize the modifications to the code, limiting them to those absolutely necessary to process Modula-2.

Very few modifications were required to transform TANGLE into MANGLE. Many more changes had to be made to WEAVE to support Modula-2, since WEAVE has to know enough about the language to format it properly. Some changes could have been made with the built-in mechanisms of WEB, such as the formatting command

format *module* ≡ *program*

which creates a new reserved word **module** and causes it to be formatted as if it were **program**. The problem of this approach is that it has to be duplicated in every source file, putting the burden of implementing MWEB on the user rather than on the developer (myself). I decided to add the Modula-2 reserved words into the internal tables. Several new reserved words were added (**return**, **exit**, **by**, **import**, etc.) and others not needed for Modula-2 were dropped (**goto**, **label**, **downto**, **file**, and others).

The following issues surfaced during the implementation of MWEB:

- *Identifier length.* The size of an identifier had to be increased. The TANGLE limit was insufficient, since some of the standard Modula-2 library modules had identifiers far longer, and the truncated identifiers would not match. Unlike Pascal, Modula-2 does not specify a maximum identifier length; all characters in an identifier are considered significant. However, since it is difficult to use ∞ as a constant in a computer program, I just picked a number out of my hat — 31 characters maximum length, 20 for unambiguous length. It can be changed if needed. The length of reserved words also had to be increased so that words such as **definition** and **implementation** could be accommodated.
- *Comments.* All code related to comments had to be changed. While Pascal can have comments delimited by either (* *) or { }, Modula-2 uses only the former, since the braces are used elsewhere in the language definition (as set delimiters). Fortunately, this is a common and well-documented modification to TANGLE, since some of the more primitive Pascal systems have the same restriction. On the other hand, Modula-2 allows nested comments, so the comment-handling code in MANGLE could be simplified (the comment delimiters for the inner nest levels no longer have to be converted to [ ] for the program to compile). The metacomment delimiters are still @{ and @}, although they are converted to (* *) when output.
- *Case sensitivity.* The automatic forcing of everything to uppercase by MANGLE was a potential problem, since Modula-2 is case-sensitive. This mechanism could not be disabled, because the Modula-2 reserved words *do* have to be uppercase and MANGLE cannot differentiate reserved words from any other identifiers. I considered giving MANGLE a reserved word table like that of MWEAVE, but that was a more radical change to the code of TANGLE than I had planned. I finally decided this was a non-problem, since all occurrences of an identifier, definition and references alike, are forced to uppercase on an equal basis. If definition modules, implementation modules, and client modules are all MANGLED, all instances of the identifier will still match. This automatic forcing to uppercase removes the requirement in Modula-2 of reserved words being in uppercase in the source. As described above, the uppercase words are for readability, but the bold font used by MWEB is much more readable. Leaving MANGLE's uppercase mechanism intact disables the ability of Modula-2 to have multiple identifiers in a program differing only in case, (*junk*, *Junk*, and *JUNK*), but I consider this a poor practice anyway. (I will stop just short of saying that anyone who does it deserves whatever happens to them.) The only real problem with the uppercase characters occurs with imported modules which were not generated with the MWEB system (such as the library modules supplied with the compiler). For identifiers such as these, which *must* contain lower or mixed case, the WEB command to "pass through" Pascal

code without modification (@=*verbatim text*@>) must be used. For example:

```
from @= InOut @> import \\ @= WriteString @>;
```

Some predefined identifiers are all uppercase to begin with, such as the primary library module SYSTEM or the increment instruction INC. These can be left alone.

- *Vertical bar character.* The vertical bar character (|) had to be specially handled, since it is used by both Modula-2 and WEB for different purposes. WEB uses it to delimit Pascal code embedded within TEX code, such as

```
The value of |good_stuff|
should be output only if
|buffer_index<=47|,
otherwise ...
```

while Modula-2 uses it to mark the end of the statement sequence following a case label (except for the last one), as in

```
case junk of
1: r := 10;
   m := 60   |
2: k := 7    |
3: m := 6
end;
```

A true conflict between the two usages is unlikely, because Pascal code within TEX code usually consists of short expressions or simple variable names rather than compound statements such as **case**. MWEAVE has been modified to identify the usage of the vertical bar by context. It will use the Modula-2 version in the code part of a section and the WEB version within TEX code (including module names and comments in the code section). If for some reason a **case** statement is needed within TEX code, two adjacent bar characters (||) are used to represent the Modula-2 case separator and are compressed by MWEAVE into an internal character which is output as the regular vertical bar character.

- *Underline character.* The usage of the underline character in identifiers, absent from the Modula-2 language definition as it is from standard Pascal, is provided by MWEB. I agree with Donald Knuth that

identifiers_several_words_long are much more readable than IdentifiersSeveralWordsLong, which is Wirth's approach. MANGLE removes the underlines before passing the program to the compiler, like TANGLE does.

- *Other special characters.* Modula-2 adds some new special characters to optionally replace tokens which require a two-character combination or a reserved word in Pascal (# for <>, ~ for **not**, and & for **and**). Modifying MANGLE and MWEAVE to handle & and ~ was no problem, but # is already used by the WEB system for macro parameters. For example, the two definitions

```
@d test(#)==m[#] <> x[j+#]
@d test(#)==m[#] #  x[j+#]
```

are logically equivalent from a Modula-2 standpoint, but the output generated by the regular TANGLE and WEAVE for the second would not be what the programmer expected. The parsers of the two programs would not be able to differentiate between the middle #, which means $\neq$, and those in the array index expressions, which are intended to be replaced by the macro parameter. To resolve this ambiguity, MANGLE and MWEAVE have been modified to accept the Modula-2 version of # anywhere in a WEB program *except* within a macro definition, where # will continue to represent the parameter.

Of course, the old Pascal symbols still work. These new symbols, and the modifications to handle them, are largely irrelevant when the WEB system is being used, because MWEAVE will convert them to $\neq$, $\neg$, and $\wedge$ anyway.

- *Quotes.* Modula-2 allows strings to be delimited by either single or double quotes (' or "). While this is a definite improvement, it does conflict with WEB, in which single quotes delimit regular strings, while double quotes identify strings destined for the "string pool", a special WEB mechanism whereby the strings so designated are written to a separate text file to be read at run-time. Rather than disable the string pool, I reluctantly decided that the user would just have to continue using single quotes as in Pascal.
- *The macro package.* Surprisingly few modifications were required to the WEB macro package, WEBMAC.TEX. In fact, I decided not to modify it at all. A new file, MWEBMAC.TEX, inputs the original WEBMAC.TEX, then redefines one macro and adds one new one. The comment macro \C{...} was redefined to generate (* *) instead of { }, and \VB was added to generate the vertical bar character. Not counting blank lines, MWEBMAC.TEX is only five lines long.

The sample program provided with this article, SCANTEX, actually performs a useful function. It scans a TEX file generated by WEAVE (or MWEAVE, of

course) and copies only the sections which have been modified by a change file to a new TEX file, resulting in an abbreviated program listing containing only the changes. The unchanged sections are not copied, nor are the cross-references, the section names, or the title page, which includes the table of contents. Typically, a WEB program running on a wide range of machines (such as TEX itself) has a great number of change files applied to it. For the most part, the main portion of the program is identical in all implementations and certain sections, containing "system dependencies", are different for each one. Since WEAVE generates a complete listing every time it is run, and a program the size of WEAVE or TANGLE runs to about a hundred pages (and that is small compared to TEX or METAFONT), a lot of paper is consumed printing several large listings that are essentially the same. Since writing SCANTEX, I have adopted the practice of printing one full listing generated from the pure WEB source (the way it comes from Stanford, with no change files applied), followed by an abbreviated listing generated by SCANTEX for each change file applied to that program. (In fact, in some cases I print *only* the changed sections, referring to published versions of the pure WEB source rather than printing it myself. In the case of TEX, I refer to the book *TEX: The Program*; most of the other Stanford-developed programs also exist as bound documents (available from Maria Code).

Experienced WEB users may wonder why I went to the trouble of writing a program to duplicate a function already provided by the WEB system itself, since the suppression of unchanged sections can be accomplished by placing

```
\let\maybe=\iffalse
```

into the limbo section of the WEB file. The reasons were:

1. SCANTEX does not print the index. Since the index contains entries for the full listing rather than just the abbreviated one, a program the size of TEX can have an index that is much longer than the rest of the listing.
2. Since SCANTEX is an external program, neither the WEB file nor the changefile need be modified to turn suppression on or off.
3. If the TEX file is to be saved, the reduced version generated by SCANTEX takes up much less disk space.
4. I was unaware that the builtin mechanism existed when I wrote SCANTEX (the real reason). Since it is buried deep in Appendix G of the WEB manual, it is easy to miss.

As can be seen from the SCANTEX listing, MWEB is not *that* different; on first glance, it could be mistaken for standard WEB. Closer inspection would reveal the differences in reserved words, in comments, and in compound statements. Note the use of the **elsif** statement. The boxes around words such as "WriteString" are an unforeseen side effect of use of the "pass through" WEB command described above to keep selected words from being forced to uppercase. While startling, it does point out which identifiers require special treatment. I highly recommend using the approach taken in SCANTEX: define simple macros equivalent to each of these external identifiers and use the macro everywhere else in the program, including the **import** statement. This isolates the boxes to the section containing the definitions rather than sprinkling them throughout the program.

Hopefully, MANGLE, MWEAVE, MWEBMAC.TEX, SCANTEX, and a few other sample MWEB programs should be available from Stanford on the regular distribution tape by the time this reaches print. The WEB files and the VAX implementation files should be available from Stanford and additionally from Kellerman and Smith. For people who have absolutely no way of reading a magnetic tape, the IBM PC version is available from me on PC floppies for a handling fee. Additionally, the original TANGLE and WEAVE, the WEBMERGE program described elsewhere in this issue (page 117), and several of the TEX and METAFONT utility programs (sometimes referred to as TEXware and METAFONTware) are also available on floppy. All of these have change files targeted for Microsoft Pascal running under MS-DOS on the IBM PC, which is my development system. As far as other target computers are concerned, MANGLE and MWEAVE are implemented as standard change files applied to TANGLE and WEAVE, so they can be merged with the current implementation-specific change files. WEBMERGE can be used for this purpose. If you have TANGLE and WEAVE running, you should have no trouble with MANGLE and MWEAVE.