The WEAVE processor

(Version 4.5)
1* Introduction. This program converts a WEB file to a TeX file. It was written by D. E. Knuth in October, 1981; a somewhat similar SAIL program had been developed in March, 1979, although the earlier program used a top-down parsing method that is quite different from the present scheme.

The code uses a few features of the local Pascal compiler that may need to be changed in other installations:

1) Case statements have a default.
2) Input-output routines may need to be adapted for use with a particular character set and/or for printing messages on the user’s terminal.

These features are also present in the Pascal version of TeX, where they are used in a similar (but more complex) way. System-dependent portions of WEAVE can be identified by looking at the entries for ‘system dependencies’ in the index below.

The “banner line” defined here should be changed whenever WEAVE is modified.

\begin{verbatim}
define my_name \equiv \texttt{weave}
define banner \equiv \texttt{This is WEAVE, Version 4.5}
\end{verbatim}

2* The program begins with a fairly normal header, made up of pieces that will mostly be filled in later. The WEB input comes from files \textit{web.file} and \textit{change.file}, and the TeX output goes to file \textit{tex.file}.

If it is necessary to abort the job because of a fatal error, the program calls the \texttt{\textit{jump_out}} procedure.

\begin{verbatim}
⟨ Compiler directives 4 ⟩
program WEAVE(web_file, change_file, tex_file);
const ⟨ Constants in the outer block 8* ⟩
type ⟨ Types in the outer block 11 ⟩
var ⟨ Globals in the outer block 9 ⟩
  ⟨ Define parse_arguments 264* ⟩
  ⟨ Error handling procedures 30 ⟩
procedure initialize;
  var ⟨ Local variables for initialization 16 ⟩
  begin kpse_set_program_name(argv[0], my_name); parse_arguments; ⟨ Set initial values 10 ⟩
  end;
8* The following parameters are set big enough to handle TeX, so they should be sufficient for most applications of WEAVE.
  ⟨ Constants in the outer block 8* ⟩ \equiv
    max_bytes = 65535; \{ 1/\textit{ww} times the number of bytes in identifiers, index entries, and module names; must be less than 65536 \}
    max_names = 10239; \{ number of identifiers, index entries, and module names; must be less than 10240 \}
    max_modules = 4000; \{ greater than the total number of modules \}
    hash_size = 8501; \{ should be prime \}
    buf_size = 1000; \{ maximum length of input line \}
    longest_name = 10000; \{ module names shouldn’t be longer than this \}
    long_buf_size = buf_size + longest_name; \{ C arithmetic in Pascal constant \}
    line_length = 80; \{ lines of TeX output have at most this many characters, should be less than 256 \}
    max.refs = 65535; \{ number of cross references; must be less than 65536 \}
    max.toks = 65535; \{ number of symbols in Pascal texts being parsed; must be less than 65536 \}
    max.texts = 10239; \{ number of phrases in Pascal texts being parsed; must be less than 10240 \}
    max.scraps = 10000; \{ number of tokens in Pascal texts being parsed \}
    stack_size = 2000; \{ number of simultaneous output levels \}
\end{verbatim}

This code is used in section 2*.
The original Pascal compiler was designed in the late 60s, when six-bit character sets were common, so it did not make provision for lowercase letters. Nowadays, of course, we need to deal with both capital and small letters in a convenient way, so WEB assumes that it is being used with a Pascal whose character set contains at least the characters of standard ASCII as listed above. Some Pascal compilers use the original name char for the data type associated with the characters in text files, while other Pascals consider char to be a 64-element subrange of a larger data type that has some other name.

In order to accommodate this difference, we shall use the name text_char to stand for the data type of the characters in the input and output files. We shall also assume that text_char consists of the elements chr(first_text_char) through chr(last_text_char), inclusive. The following definitions should be adjusted if necessary.

\[
\begin{align*}
\text{define } \text{text_char} & \equiv \text{ASCII code} \quad \{ \text{the data type of characters in text files} \} \\
\text{define } \text{first_text_char} & = 0 \quad \{ \text{ordinal number of the smallest element of text_char} \} \\
\text{define } \text{last_text_char} & = 255 \quad \{ \text{ordinal number of the largest element of text_char} \}
\end{align*}
\]

Text file = packed file of text_char;

Here now is the system-dependent part of the character set. If WEB is being implemented on a garden-variety Pascal for which only standard ASCII codes will appear in the input and output files, you don’t need to make any changes here. But if you have, for example, an extended character set like the one in Appendix C of The TEXbook, the first line of code in this module should be changed to

\[
\text{for } i \leftarrow 1 \text{ to } ^{\prime} 37 \text{ do } \text{xchr}[i] \leftarrow \text{chr}(i);
\]

WEB’s character set is essentially identical to TEX’s, even with respect to characters less than ‘40.

Changes to the present module will make WEB more friendly on computers that have an extended character set, so that one can type things like ≠ instead of <>. If you have an extended set of characters that are easily incorporated into text files, you can assign codes arbitrarily here, giving an xchr equivalent to whatever characters the users of WEB are allowed to have in their input files, provided that unsuitable characters do not correspond to special codes like carriage_return that are listed above.

(The present file WEAVE.WEB does not contain any of the non-ASCII characters, because it is intended to be used with all implementations of WEB. It was originally created on a Stanford system that has a convenient extended character set, then “sanitized” by applying another program that transliterated all of the non-standard characters into standard equivalents.)

\[
\begin{align*}
\text{for } i \leftarrow 1 \text{ to } ^{\prime} 37 \text{ do } \text{xchr}[i] \leftarrow \text{chr}(i); \\
\text{for } i \leftarrow 200 \text{ to } 377 \text{ do } \text{xchr}[i] \leftarrow \text{chr}(i);
\end{align*}
\]
Terminal output is done by writing on file term_out, which is assumed to consist of characters of type text_char:

```
define term_out ≡ stdout
define print(#) ≡ write(term_out, #) {‘print’ means write on the terminal}
define print_ln(#) ≡ write_ln(term_out, #) {‘print’ and then start new line}
define new_line ≡ write_ln(term_out) {start new line}
define print_nl(#) ≡ {
  print information starting on a new line
  begin new_line; print(#);
end}
```

Different systems have different ways of specifying that the output on a certain file will appear on the user’s terminal.

\( \langle \text{Set initial values } 10 \rangle +≡ \)

\{ nothing need be done \}

The update_terminal procedure is called when we want to make sure that everything we have output to the terminal so far has actually left the computer’s internal buffers and been sent.

```
define update_terminal ≡ fflush(term_out) {empty the terminal output buffer}
```

The following code opens the input files. This is called after the filename variables have been set appropriately.

```
procedure open_input; {prepare to read web_file and change_file}
  begin web_file ← kpse_open_file(web_name, kpse_web_format);
    if chg_name then change_file ← kpse_open_file(chg_name, kpse_web_format);
  end;
```

The following code opens tex_file. Since this file was listed in the program header, we assume that the Pascal runtime system has checked that a suitable external file name has been given.

\( \langle \text{Set initial values } 10 \rangle +≡ \)

\( \text{rewrite(tex_file, tex_name);} \)
The `input ln` procedure brings the next line of input from the specified file into the `buffer` array and returns the value `true`, unless the file has already been entirely read, in which case it returns `false`. The conventions of `TeX` are followed; i.e., `ASCII_code` numbers representing the next line of the file are input into `buffer[0]`, `buffer[1]`, ..., `buffer[limit - 1]`; trailing blanks are ignored; and the global variable `limit` is set to the length of the line. The value of `limit` must be strictly less than `bufsize`.

We assume that none of the `ASCII_code` values of `buffer[j]` for `0 ≤ j < limit` is equal to 0, '177, `line_feed`, `form_feed`, or `carriage_return`. Since `bufsize` is strictly less than `long_bufsize`, some of `WEAVE`'s routines use the fact that it is safe to refer to `buffer[limit + 2]` without overstepping the bounds of the array.

```plaintext
function input ln(var f : text_file): boolean; {inputs a line or returns false}
  var final_limit: 0..bufsize;  {limit without trailing blanks}
  begin limit ← 0; final_limit ← 0;
  if eof(f) then input ln ← false
  else begin while ¬eoln(f) do
    begin buffer[limit] ← xord[getc(f)]; incr(limit);
    if buffer[limit - 1] ≠ "\n" then final_limit ← limit;
    if limit = bufsize then
      begin while ¬eoln(f) do vgetc(f);
        decr(limit); {keep buffer[bufsize] empty}
      if final_limit > limit then final_limit ← limit;
        print nl("!Input\nline too long"); loc ← 0; error;
      end;
    end;
    read ln(f); limit ← final_limit; input ln ← true;
  end;
end;
```
The `jump_out` procedure just cuts across all active procedure levels and jumps out of the program. It is used when no recovery from a particular error has been provided.

```plaintext
define fatal_error(#) ≡
    begin
        new_line; write(stderr,#); error; mark_fatal; jump_out;
    end
end
```

```plaintext
procedure jump_out;
    begin
        stat ⟨Print statistics about memory usage 262⟩; tats
        ⟨Print the job history 263⟩;
        new_line;
        if (history ≠ spotless) ∧ (history ≠ harmless_message) then uexit(1)
        else uexit(0);
    end;
```
`WEAVE` changes for C DATA STRUCTURES

WEAVE has been designed to avoid the need for indices that are more than sixteen bits wide, so that it can be used on most computers. But there are programs that need more than 65536 bytes; \TeX is one of these (and the pdf\TeX variant even requires more than twice that amount when its “final” change file is applied). To get around this problem, a slight complication has been added to the data structures: byte\_\_mem is a two-dimensional array, whose first index is either 0, 1 or 2. (For generality, the first index is actually allowed to run between 0 and \(ww - 1\), where \(ww\) is defined to be 3; the program will work for any positive value of \(ww\), and it can be simplified in obvious ways if \(ww = 1\).)

```latex
\begin{verbatim}
define ww = 3  \{ we multiply the byte capacity by approximately this amount \}
\end{verbatim}
```

<table>
<thead>
<tr>
<th>(Globals in the outer block 9)</th>
<th>+ \equiv</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte__mem: packed array [0 .. (ww - 1), 0 .. max_bytes] of ASCII_code;  { characters of names }</td>
<td></td>
</tr>
<tr>
<td>byte__start: array [0 .. max_names] of sixteen_bits;  { directory into byte__mem }</td>
<td></td>
</tr>
<tr>
<td>link: array [0 .. max_names] of sixteen_bits;  { hash table or tree links }</td>
<td></td>
</tr>
<tr>
<td>ilk: array [0 .. max_names] of sixteen_bits;  { type codes or tree links }</td>
<td></td>
</tr>
<tr>
<td>xref: array [0 .. max_names] of sixteen_bits;  { heads of cross-reference lists }</td>
<td></td>
</tr>
</tbody>
</table>

50*  A new cross reference for an identifier is formed by calling \texttt{new\_xref}, which discards duplicate entries and ignores non-underlined references to one-letter identifiers or Pascal’s reserved words.

If the user has sent the \texttt{no\_xref} flag (the -x option of the command line), then it is unnecessary to keep track of cross references for identifiers. If one were careful, one could probably make more changes around module 100 to avoid a lot of identifier looking up.

```latex
\begin{verbatim}
define append\_xref(#) \equiv
  if xref\_ptr = max\_refs then overflow(\textquoteleft\text{cross\_reference}\textquoteleft)
  else begin incr(xref\_ptr); num(xref\_ptr) \leftarrow \#
  end
\end{verbatim}
```

```latex
\begin{verbatim}
procedure new\_xref(p: name\_pointer);
  label exit;
  var q: xref\_number;  \{ pointer to previous cross-reference \}
  m, n: sixteen\_bits;  \{ new and previous cross-reference value \}
  begin if no\_xref then return;
  if (reserved(p) \lor (byte\_\_start[p] + 1 = byte\_\_start[p + ww])) \land (xref\_switch = 0) then return;
  m \leftarrow module\_count + xref\_switch; xref\_switch \leftarrow 0; q \leftarrow xref[p];
  if q > 0 then
    begin n \leftarrow num(q);
      if (n = m) \lor (n = m + def\_flag) then return
      else if m = n + def\_flag then
        begin num(q) \leftarrow m; return;
      end;
    end;
    append\_xref(m); xlink(xref\_ptr) \leftarrow q; xref[p] \leftarrow xref\_ptr;
  exit: end;
\end{verbatim}
```
The `reduce` macro used in our code for productions actually calls on a procedure named `red`, which makes the appropriate changes to the scrap list.

```plaintext
procedure red(j: sixteen_bits; k: eight_bits; c: eight_bits; d: integer);
  var i: 0..max_scraps; { index into scrap memory }
  begin cat[j] ← c; trans[j] ← text_ptr; freeze_text;
    if k > 1 then
      begin for i ← j + k to lo_ptr do
          begin cat[i + 1] ← cat[i]; trans[i + 1] ← trans[i];
          end;
        lo_ptr ← lo_ptr − k + 1;
      end;
      ⟨ Change pp to max(scrap_base, pp + d) ⟩
  end;
```

This code is used in sections 172* and 174.
Phase three processing. We are nearly finished! WEAVE's only remaining task is to write out the index, after sorting the identifiers and index entries.

If the user has set the no_xref flag (the -x option on the command line), just finish off the page, omitting the index, module name list, and table of contents.

\begin{verbatim}
Phase III: Output the cross-reference index
\end{verbatim}

\begin{verbatim}
if no_xref then
  begin finish_line; out("\n"); out5("v")("f")("i")("l"); out4("n")("d"); finish_line;
else begin phase_three ← true; print_nl(´Writing_\#\#\#\#index...´);
  if change_exists then
    begin finish_line; ⟨Tell about changed modules⟩;
  end;
  finish_line; out5("i")("n"); finish_line; ⟨Do the first pass of sorting⟩;
  ⟨Sort and output the index⟩;
  out4("c")("n"); finish_line; ⟨Output all the module names⟩;
  end;
print(´Done.`);
\end{verbatim}

This code is used in section 261*. 

Debugging. The Pascal debugger with which WEAVE was developed allows breakpoints to be set, and variables can be read and changed, but procedures cannot be executed. Therefore a `debug_help` procedure has been inserted in the main loops of each phase of the program; when `ddt` and `dd` are set to appropriate values, symbolic printouts of various tables will appear.

The idea is to set a breakpoint inside the `debug_help` routine, at the place of ‘breakpoint:’ below. Then when `debug_help` is to be activated, set `trouble_shooting` equal to `true`. The `debug_help` routine will prompt you for values of `ddt` and `dd`, discontinuing this when `ddt ≤ 0`; thus you type `2n + 1` integers, ending with zero or a negative number. Then control either passes to the breakpoint, allowing you to look at and/or change variables (if you typed zero), or to exit the routine (if you typed a negative value).

Another global variable, `debug_cycle`, can be used to skip silently past calls on `debug_help`. If you set `debug_cycle > 1`, the program stops only every `debug_cycle` times `debug_help` is called; however, any error stop will set `debug_cycle` to zero.

```
define term_in ≡ stdin
         ⟨Globals in the outer block 9⟩ +≡
         debug trouble_shooting: boolean;  { is `debug_help` wanted? }
         ddt: integer;  { operation code for the `debug_help` routine }
         dd: integer;  { operand in procedures performed by `debug_help` }
         debug_cycle: integer;  { threshold for `debug_help` stopping }
         debug_skipped: integer;  { we have skipped this many `debug_help` calls }

         gubed
```

The debugging routine needs to read from the user’s terminal.

```
         ⟨Set initial values 10⟩ +≡
         debug trouble_shooting ← true; debug_cycle ← 1; debug_skipped ← 0; tracing ← 0;
         trouble_shooting ← false; debug_cycle ← 99999;  { use these when it almost works }

         gubed
```
261* The main program. Let’s put it all together now: WEAVE starts and ends here.

The main procedure has been split into three sub-procedures in order to keep certain Pascal compilers from overflowing their capacity.

procedure Phase_I;
  begin ⟨Phase I: Read all the user’s text and store the cross references⟩;
  end;

procedure Phase_II;
  begin ⟨Phase II: Read all the text again and translate it to \TeX form⟩;
  end;

  begin initialize; { beginning of the main program }
    print(banner); { print a “banner line” }
    print_{ln}(version_string); { Store all the reserved words }
    Phase_I; Phase_II;
  ⟨Phase III: Output the cross-reference index⟩;
  ⟨Check that all changes have been read⟩;
  jump_out;
  end.
264* System-dependent changes. Parse a Unix-style command line.

```c

#define argument_is(#) (strcmp(long_options[option_index].name, #) == 0)

// Define parse_arguments 264*

procedure parse_arguments;

const n_options = 4;  // Pascal won't count array lengths for us.

var long_options: array [0 .. n_options] of getopt_struct;

getopt_return_val: integer; option_index: c_int_type; current_option: 0 .. n_options;

begin (Define the option table 265*);

repeat getopt_return_val ← getopt_long_only(argc, argv, ``, long_options, address_of(option_index));

if getopt_return_val = -1 then
  begin do_nothing;  // End of arguments; we exit the loop below. }
  end

else if getopt_return_val = `?` then
  begin usage(my_name);
  end

else if argument_is(`help`) then
  begin usage_help(WEAVE_HELP, nil);
  end

else if argument_is(`version`) then
  begin print_version_and_exit(banner, nil, `D.E.Knuth`, nil);
  end

end;  // Else it was a flag; getopt has already done the assignment. }

until getopt_return_val = -1;  // Now optind is the index of first non-option on the command line. }

if (optind + 1 > argc) ∨ (optind + 3 < argc) then
  begin writeLn(stderr, my_name, `:Need one to three file arguments.`); usage(my_name);
  end;  // Supply ".web" and ".ch" extensions if necessary. }

web_name ← extend_filename(cmdline(optind), `web`);

if optind + 2 ≤ argc then
  begin if strcmp(char_to_string(`-`), cmdline(optind + 1)) ≠ 0 then
    chg_name ← extend_filename(cmdline(optind + 1), `ch`);
  end;  // Change ".web" to ".tex" and use the current directory. }

if optind + 3 = argc then
tex_name ← extend_filename(cmdline(optind + 2), `tex`)
else
tex_name ← basename_change_suffix(web_name, `.web`, `.tex`);

end;
```

This code is used in section 2*.

265* Here are the options we allow. The first is one of the standard GNU options.

```c

// Define the option table 265*}

current_option ← 0; long_options[current_option].name ← `help`;

long_options[current_option].has_arg ← 0; long_options[current_option].flag ← 0;

long_options[current_option].val ← 0; incr(current_option);

// See also sections 266*, 267*, and 269*.

This code is used in section 264*.

266* Another of the standard options.

```c

// Define the option table 265* +

long_options[current_option].name ← `version`; long_options[current_option].has_arg ← 0;

long_options[current_option].flag ← 0; long_options[current_option].val ← 0; incr(current_option);
```
§267* Omit cross-referencing?
\(\langle\text{Define the option table}\ 265*\rangle\) +≡
\[
\begin{align*}
\text{long_options}[\text{current_option}].\text{name} & \leftarrow \text{char_to_string(}'x'\text{'})\;\text{;} \; \\
\text{long_options}[\text{current_option}].\text{has_arg} & \leftarrow 0\;\text{;} \; \\
\text{long_options}[\text{current_option}].\text{flag} & \leftarrow \text{address_of(}\text{no_xref}\text{)}\;\text{;} \; \\
\text{long_options}[\text{current_option}].\text{val} & \leftarrow 1\;\text{;} \\
\text{inc}r(\text{current_option})\;\text{;}
\end{align*}
\]

§268* (Globals in the outer block 9) +≡
\[
\text{no_xref}:\;\text{c_int_type}\
\]

§269* An element with all zeros always ends the list.
\(\langle\text{Define the option table}\ 265*\rangle\) +≡
\[
\begin{align*}
\text{long_options}[\text{current_option}].\text{name} & \leftarrow 0\;\text{;} \; \\
\text{long_options}[\text{current_option}].\text{has_arg} & \leftarrow 0\;\text{;} \\
\text{long_options}[\text{current_option}].\text{flag} & \leftarrow 0\;\text{;} \; \\
\text{long_options}[\text{current_option}].\text{val} & \leftarrow 0\;\text{;}
\end{align*}
\]

§270* Global filenames.
\(\langle\text{Globals in the outer block}\ 9\rangle\) +≡
\[
\text{web_name, chg_name, tex_name}:\;\text{const}\_\text{c_string}\
\]
271* Index. If you have read and understood the code for Phase III above, you know what is in this index and how it got here. All modules in which an identifier is used are listed with that identifier, except that reserved words are indexed only when they appear in format definitions, and the appearances of identifiers in module names are not indexed. Underlined entries correspond to where the identifier was declared. Error messages, control sequences put into the output, and a few other things like “recursion” are indexed here too.

The following sections were changed by the change file: 1, 2, 8, 12, 17, 20, 21, 22, 24, 26, 28, 33, 37, 50, 172, 173, 239, 258, 259, 261, 264, 265, 266, 267, 268, 269, 270, 271.

\-help: 265*
\-version: 266*
\-x: 267*
\(): 186.
\*(): 130.
\,: 162, 163, 186.
\:. 189, 253.
\:: 252, 256.
\=: 189.
\[(): 254.
\[@(): 186, 189, 193.
\@(): 186, 189.
\$: 186, 189.
\%(): 186, 189.
\&(): 189, 209, 253.
\^(): 189.
\^(): 189, 209, 253.
\^(): 186, 189.
\^(): 189.
\{(): 189.
\}(): 189.
\~(): 189, 193.
\[(): 186.
\[(): 209, 253.
\_: 131, 189.
\A(): 236.
\As(): 237.
\ast(): 186.
\B(): 186.
\C(): 198.
\con(): 239*
\D(): 227.
\E(): 186.
\ET(): 237.
\ETs(): 237.
\F(): 228.
\fi(): 238.
\fin(): 239*
\G(): 188.
\H(): 196, 224.
\I(): 188.
\in(): 186.
\input webmac: 124.
\inx: 239*
app3:  148, 158, 164.
arq:   264*
argument_is:  264*
arqv:   2*264*
array_like:  42, 64, 191, 192.
ASCII code:  11, 86.
ASCII_code:  11, 12, 13, 27, 28*37*65, 73, 87, 89, 91, 121, 127, 132, 136, 208, 242, 246, 247, 249.
b:     122, 208.
backup:  141, 142, 143, 147, 160, 167, 192, 193, 208, 231.
banner: 1*261*264*
basename_change_suffix:  264*
begint:  3.
begin_comment:  86, 87, 97, 186, 222.
begin_like:  42, 64, 193.
begin_Pascal:  86, 87, 117, 229, 230.
begintag:  140, 142, 143, 150, 152, 164, 170, 193.
big_cancel:  141, 142, 147, 186, 208, 212.
big_force:  141, 142, 147, 186, 208, 212, 226.
big_line_break:  86, 87, 186, 222.
blink:   242, 243, 250, 251, 252.
boolean:  28*29, 45, 71, 74, 93, 122, 143, 258*
breakout: 125, 126, 127.
break_space:  141, 143, 147, 152, 155, 156, 160, 164, 168, 170, 200, 208, 211, 212.
breakpoint:  258*260.
bucket:  242, 243, 249, 251.
bufsize:  8*28*73, 74, 75, 79, 123.
buffer:  27, 28*31, 32, 55, 58, 59, 61, 62, 63, 74, 76, 78, 79, 80, 81, 82, 84, 85, 89, 90, 91, 92, 93, 95, 97, 98, 99, 100, 103, 104, 106, 107, 110, 123, 132, 133, 134, 135, 136, 137, 179, 182, 183, 189, 190, 196, 208, 214, 216, 217, 221, 223, 224, 260.
c_int_type:  264*268*.
cancel:  141, 142, 143, 147, 153, 155, 156, 159, 160, 162, 163, 164, 165, 193, 197, 200, 208, 211, 212.
carriage_return:  15, 17*28*.
carryover:  122.
case_head:  140, 143, 149, 153, 154, 162, 163, 165.
cur_name:  63, 206, 209, 213, 214, 242, 243, 251, 252, 253, 255.
cur_state:  202, 204, 205.
cur_tok:  201, 202, 204, 205, 206.
cur_val:  244, 254.
current_option:  264* 265* 266* 267* 269*.
d:  95, 127, 172* 174, 249.
 dd:  258* 260.
ddt:  258* 260.
debug_cycle:  31, 258* 259* 260.
debug_help:  30, 31, 95, 206, 252, 258* 260.
debug_skipped:  31, 258* 259* 260.
decr:  6, 28* 92, 98, 103, 122, 127, 130, 135, 137, 138, 205, 251, 252.
definition:  86, 87, 115, 225.
depth:  244, 249.
dig:  129, 130.
div_like:  42, 64, 191, 192.
do_like:  42, 64, 191, 192.
do_nothing:  6, 95, 113, 149, 150, 186, 189, 222, 253, 264*.
done:  5, 75, 76, 90, 91, 92, 95, 103, 104, 122, 134, 135, 136, 137, 175, 179, 236, 237.
Double @ required:  133.
Double @ should be used:  189.
double_dot:  86, 97, 186.
doucs_cases:  183, 185, 186.
else:  7.
else_like:  42, 64, 193.
else_if:  140, 142, 143, 149, 156, 193.
emit_space_if_needed:  219, 225, 230.
end:  3, 7.
end_comment:  86, 87, 97, 186, 222.
end_field:  201, 202.
end_like:  42, 64, 193.
end_translation:  141, 147, 201, 207, 208, 212.
endcases:  7.
eof:  28*.
eoln:  28*.
equal:  66, 67, 68.
equivalence_sign:  15, 97, 116, 188, 228, 231.
error:  28* 31, 33*.
error_message:  9, 263.
exit:  5, 6, 50* 74, 75, 79, 89, 111, 123, 127, 132, 183, 208, 236, 260.
exp:  140, 142, 143, 149, 186.
exponent:  93, 98, 186.
extend_filename:  264*.
extension:  66, 68, 69.
Extra } :  95.
Extra @> :  87.
f:  28*.
false:  28* 29, 72, 73, 74, 79, 81, 84, 94, 96, 109, 122, 123, 127, 218, 238, 259*.
fatal_error:  33* 34, 35.
fatal_message:  9, 263.
flush:  22*.
final_limit:  28*.
finish_line:  123, 124, 132, 134, 135, 212, 218, 226, 236, 238, 259* 254, 256.
first_text_char:  12* 18.
first_xref:  234, 235.
five_cases:  149, 150.
flag:  236, 237, 265* 266* 267* 269*.
flush_buffer:  122, 123, 127, 128, 218, 238.
footnote:  233, 236, 256.
for_like:  42, 64, 191, 192.
form_feed:  15, 28*.
format:  86, 87, 186.
forward:  30, 207.
found:  5, 58, 60, 61, 66, 95, 96, 122, 148, 149, 150, 151, 175, 179, 208, 216.
freeze_text:  171, 172* 180, 184, 195, 198, 208.
get_line:  71, 82, 89, 90, 91, 95, 103, 123, 132, 134, 136.
get_output:  206, 207, 208, 211, 212.
getc:  28*.
getopt:  264*.
getopt_long_only:  264*.
getopt_return_val:  264*.
getopt_struct:  264*.
goto_like:  42, 64, 191, 192.
greater:  66, 68, 69.
greater_or_equal:  15, 97, 188.
Illegal control code...

harmless

hash:

hash_size:

head:

hex:

history:

Hmm... n of the preceding...: 80.

i:

id_first:

id_loc:

id_lookup:

identifier:

id2:

id3:

id4:

id5:

id6:

id7:

id8:

id9:

if like:

if_module_start_then_make_change_pending:

ignore:

ii:

ilk:

Illegal use of @...: 215.

Improper format definition: 228.

Improper macro definition: 227.

Incompatible section names: 66.

incr:

indent:

infinity:

init_stack:

initialize:

inner:

inner_tok_flag:

Input ended in mid-comment: 136.

Input ended in section name: 103.

Input line too long: 28*

input_has Ended:

input_ln:

integer:

intercal_like:

intro:

k:

kpse_open_file:

kpse_set_program_name:

kpse_web_format:

l:

last_text_char:

lbrace:

left_arrow:

length:

less:

less_or_equal:

lhs:

limit:

line:

Line had to be broken: 128.

line_break:

line_feed:

line_length:

lines_dont_match:

link:

link:

lo_ptr:

loc:

long_buf_size:

long_options:

longest_name:

loop:

loop_like:

m:
make_output: 207, 208, 213, 226, 256.
mark_error: 9, 31, 215, 216.
mark_fatal: 9, 33*
meth: 139, 140, 142, 143, 148, 150, 151, 158,
160, 162, 163, 167, 170, 179, 180, 186, 188,
192, 227, 228, 231.
meth_bin: 141, 142, 147, 192, 208, 210.
meth_break: 86, 87, 186, 222.
meth_op: 141, 143, 147, 162, 163, 208.
meth_rel: 141, 142, 146, 147, 192, 208, 210, 231.
max_bytes: 8*37*, 39, 44, 58, 62, 66, 67, 69,
131, 208, 244.
max_modules: 8*45, 46, 110, 240.
maxrefs: 8*47, 50*,
max_scr_pnl: 144, 145, 187, 197, 199, 226, 262.
max_scraps: 8*144, 172*, 174, 178, 179, 187,
197, 199, 244.
max_sort_pnl: 244, 245, 249, 262.
max_sorts: 244, 249.
max_stack_pnl: 202, 203, 204, 262.
max_text: 8*52, 175, 179, 187, 199.
max_tok: 53, 54, 175, 179, 199, 207, 226, 262.
max_toks: 8*53, 136, 146, 175, 179, 180, 187,
189, 198, 199.
mid_xref: 234, 235.
Missing "|" . . . : 197.
mod_check: 119, 120.
mod_flag: 146, 206, 231, 232, 256.
mod_name: 206, 208.
mod_print: 256, 257.
mod_scrap: 140, 142, 143, 149, 167, 231, 232.
mod_text: 65, 66, 67, 68, 69, 95, 101, 102, 103,
104, 105, 260.
mod_xref_switch: 46, 48, 49, 51, 117.
mode: 201, 208.
mode_field: 201, 202.
module_count: 45, 50*, 51, 71, 79, 84, 109, 110,
181, 218, 220, 221, 231, 241.
module_name: 86, 87, 93, 100, 113, 117, 222,
230, 232.
my_name: 1*2*264*
n: 50*, 79, 178.
n_options: 264*
name: 264*, 265*, 266*, 267*, 269*
Name does not match: 69.
name_pointer: 38, 39, 44, 50*, 51, 58, 63, 66,
69, 93, 111, 114, 119, 131, 144, 183, 192,
208, 229, 242, 256.
name_ptr: 38, 39, 41, 44, 58, 60, 62, 67, 262.
Never defined: <section name>: 119.
Never used: <section name>: 119.
new_line: 20*, 31, 32, 33*, 128.
new_mod_xref: 51, 117.
new_module: 86, 87, 90, 95, 134.
next_control: 108, 111, 112, 113, 115, 116, 117,
183, 185, 186, 189, 191, 197, 198, 207, 222,
225, 227, 228, 229, 230, 231, 232.
next_name: 242, 243, 251.
next_xref: 234, 235, 255.
nil: 6.
nil_like: 42, 64, 191, 192.
no_line_break: 86, 87, 186, 222.
no_underline: 86, 87, 100, 113.
no_xref: 50*, 239*, 267*, 268*
normal: 42, 58, 60, 111, 116, 191, 192, 227,
228, 253.
not_equal: 15, 97, 188.
not_found: 5.
not_sign: 15, 64, 188.
num: 46, 49, 50*, 51, 119, 213, 231, 235, 236,
237, 254.
num_field: 46, 48.
octal: 86, 87, 186, 222.
omega: 140, 142, 143, 151, 192, 193, 195.
oot: 125.
oot1: 125.
oot2: 125.
oot3: 125.
oot4: 125.
oot5: 125.
open: 139, 140, 142, 143, 150, 186.
open_input: 24*, 81.
opt: 139, 141, 142, 143, 147, 159, 163, 186,
208, 211.
optind: 264*
option_index: 264*
or_sign: 15, 64, 188.
ord: 13.
other_line: 71, 72, 81, 85.
othecases: 7.
others: 7.
out: 125, 130, 131, 133, 135, 208, 209, 210,
211, 212, 213, 214, 221, 222, 223, 224, 236,
out_buf: 121, 122, 124, 125, 126, 127, 128, 212,
226, 231, 260.
out_line: 121, 122, 124, 128, 219.
out_mod: 130, 213, 221, 237, 241, 254.
out_name: 131, 209, 253.
outdent: 141, 143, 147, 153, 155, 156, 160, 162, 163, 164, 208.
outer: 200, 201, 202, 211, 212.
outer_parse: 198, 225, 230.
output_Pascal: 207, 214, 222.
out2: 125, 130, 210, 211, 212, 213, 219, 221, 226, 237, 241, 252, 253, 254, 256.
out3: 125, 210, 223, 224, 237, 238.
out4: 125, 226, 239* 241.
out5: 125, 210, 239*.
param: 86.
pascal_arguments: 2*, 264*.
Pascal_text...didn't end: 216.
Pascal_parse: 183, 186, 192, 196, 197, 198.
Pascal_translate: 197, 198, 207.
per_cent: 122.
Phase_I: 261*.
Phase_II: 261*.
phase_three: 29, 109, 213, 239*.
pop_level: 205, 206.
prefix: 66, 68.
prime_the_change_buffer: 75, 81, 84.
print_cat: 140, 178, 181, 260.
print_in: 20*, 32, 128, 181, 261*.
print_text: 146, 260.
print_version_end_exit: 264*.
proc: 140, 142, 143, 149, 162, 163, 164, 193.
proc_like: 42, 64, 111, 193.
prod: 148, 178, 183.
production: 148.
production_end: 148.
productions, table of: 143.
pseudo_semi: 86, 87, 186, 222.
push_level: 204, 206, 208.
q: 50*, 51, 66, 69, 198, 236.
r: 51, 69, 146.
rbrace: 146.
read: 260.
read_in: 28*.
record_head: 140, 142, 143, 149, 193.
record_like: 42, 64, 193.
recursion: 119, 207, 256.
repeat_like: 42, 64, 193.
res_flag: 146, 192, 193, 206.
res_word: 206, 208, 209.
reserved: 42, 50*, 60.
reset_input: 81, 109, 218.
restart: 5, 82, 95, 100, 206.
reswitch: 5, 183, 185, 191, 208, 212.
return: 5, 6.
rewrite: 26*.
rhs: 114, 116.
roman: 42, 111, 253.
root: 43, 66, 69, 120, 257.
save_base: 197.
save_limit: 208, 214.
save_line: 219.
save_loc: 208, 214.
save_mode: 208, 212.
save_next_control: 207.
save_place: 219.
save_text_ptr: 207.
save_tok_ptr: 207.
scanning_hex: 93, 94, 95, 96, 100.
scrap_ptr: 144, 145, 176, 178, 179, 183, 184, 187, 194, 195, 197, 199, 226, 228, 244, 256.
sc0: 184, 186, 193, 194, 195, 228.
sc1: 184, 186, 189, 190, 192, 193, 196, 227, 228, 231, 232.
sc2: 184, 186, 188, 192, 193, 227, 228, 231.
sc3: 184, 186, 192, 193, 231.
sc4: 184, 186, 193.
Section name didn't end: 104.
Section name too long: 105.
semi: 139, 140, 142, 143, 149, 161, 163, 186, 194, 195, 228, 231.
set_element_sign: 15, 64, 186.
term:

temp:

system dependencies:

sixteen_bits:

skip_comment:

skip_limbo:

skip_TeX:

Sorry, x capacity exceeded:

split procedures:

special string characters:

sq:

squad:

stack:

stack_ptr:

stack_size:

stat:

stderr:

stdin:

statn:

strncmp:

string:

String constant didn't end:

string_delimiter:

sub_cases:

system dependencies:

s0:

s1:

s2:

s3:

s4:

t:

tab_mark:

tats:

temp_line:

term_in:

term_out:

terminator:

TeX string should be:

tex_file: 2* 25, 26*, 122, 124.

tex_name: 26*, 264*, 270*

TeX_string: 86, 87, 93, 100, 185, 222.

text_char: 12*, 13, 20*

text_file: 12*, 23, 25, 28*

text_pointer: 52, 53, 144, 146, 179, 197, 198, 204, 207, 226.


thin_space: 86, 87, 186, 222.

This can't happen: 34.


this_xref: 234, 235, 255.

to_like: 42, 64, 191, 192.

tok_field: 201, 202.


tok_start: 52, 53, 54, 144, 146, 171, 204.

tracing: 88, 177, 178, 181, 182, 259*

trans: 144, 148, 172*, 176, 179, 183, 184, 195, 197, 244.

translate: 149, 179, 180, 197, 226.

trouble_shooting: 95, 206, 252, 258*, 259*

ttrue: 6, 28*, 29, 71, 72, 74, 79, 81, 83, 84, 85, 93, 100, 109, 110, 122, 127, 128, 239*, 258*, 259*

typewriter: 42, 111, 253.

wexit: 33*

unbucket: 249, 250, 251.

underline: 86, 87, 100, 113.

Unknown control code: 87.

until_like: 42, 64, 193.

up_to: 95.


usage: 264*

usage_help: 264*

val: 265*, 266*, 267*, 269*

var_head: 140, 142, 143, 148, 149, 162, 163, 170, 193.

var_like: 42, 64, 111, 193.

verbatim: 86, 87, 100, 107, 185, 189.

Verbatim string didn't end: 107.

version_string: 261*

wgetc: 28*

w: 44, 58, 66, 69, 131, 208.

WEAVE: 2*

WEAVE_HELP: 264*

WEB file ended: ...: 79.
web_name: 24* 264* 270*
webmac: 124.
Where is the match...: 76, 80, 84.
wi: 40, 41.
wildcard: 42, 111, 253.
write: 20* 33*, 122, 124.
write_in: 20*, 122, 264*
ww: 8* 37*, 38, 40, 41, 44, 50* 58, 61, 62, 66, 67,
    68, 69, 131, 208, 209, 214, 243, 244, 251, 262.
xchr: 13, 14, 16, 17*, 18, 32, 44, 105, 122, 128,
    146, 147, 182, 260.
xclause: 6.
xlink_field: 46, 48.
xmem: 46, 48.
xord: 13, 16, 18, 28*
xref: 36, 37*, 46, 49, 50*, 51, 62, 67, 119, 213,
    231, 235, 243, 255.
xref_number: 47, 48, 50*, 51, 118, 234, 236.
xref_ptr: 46, 48, 49, 50*, 51, 262.
xref_roman: 86, 87, 93, 100, 111, 113, 186, 222.
xref_switch: 46, 48, 49, 50*, 93, 100, 101, 111,
    113, 115.
xref_typewriter: 86, 87, 93, 111, 113, 186, 222.
xref_wildcard: 86, 87, 93, 111, 113, 186, 222.
You can’t do that...: 222, 232.
You need an = sign...: 231.
Append a string scrap 189  
Append a TEx string scrap 190  
Append an identifier scrap 191  
Append the scrap appropriate to next_control 185  
Append terminator if not already present 194  
Cases for alpha 151  
Cases for beginning 152  
Cases for case_head 153  
Cases for casey 154  
Cases for clause 155  
Cases for cond 156  
Cases for elsei 157  
Cases for exp 158  
Cases for intro 159  
Cases for math 160  
Cases for mod_scrap 161  
Cases for open math 163  
Cases for open 162  
Cases for proc 164  
Cases for record_head 165  
Cases for semi 166  
Cases for simp 167  
Cases for stmt 168  
Cases for terminator 169  
Cases for var_head 170  
Cases involving nonstandard ASCII characters 188  
Cases that generate more than one scrap 193  
Change pp to max(scrap_base, pp+d) 173  
Check for overlong name 105  
Check that all changes have been read 85  
Check that = or \equiv follows this module name, and emit the scraps to start the module definition 231  
Clear bal and goto done 138  
Combine the irreducible scraps that remain 180  
Compare name p with current identifier, goto found if equal 61  
Compiler directives 4  
Compress two-symbol combinations like ‘:≡’ 97  
Compute the hash code h 59  
Compute the name location p 60  
Constants in the outer block 8*  
Copy a control code into the buffer 217  
Copy special things when c = "@", "\n", "\{", "\}": goto done at end 137  
Copy the Pascal text into buffer[(limit + 1) .. j] 216  
Copy up to ‘\’ or control code, goto done if finished 135  
Copy up to control code, return if finished 133  
Declaration of subprocedures for translate 150  
Declaration of the app_comment procedure 195  
Declaration of the app_octal and app_hex procedures 196  
Declaration of the easy_cases procedure 186  
Declaration of the sub_cases procedure 192  
Define the option table 265*, 266*, 267*, 269*  
Define parse_arguments 264*  
Define the option table

Used in section 183.

Used in sections 264*.

Used in section 2*.

Used in section 2*.

Used in section 185.

Used in section 185.

Used in section 185.

Used in section 185.

Used in section 183.

Used in sections 193, 193, and 193.

Used in section 150.

Used in section 150.

Used in section 149.

Used in section 149.

Used in section 149.

Used in section 149.

Used in sections 136 and 137.

Used in section 179.

Used in section 60.

Used in section 2*.

Used in section 95.

Used in section 58.

Used in section 58.

Used in section 2*.

Used in section 216.

Used in section 136.

Used in section 214.

Used in section 134.

Used in section 132.

Used in section 179.

Used in section 183.

Used in section 183.
WEAVE changes for C

NAMES OF THE SECTIONS

37

(Do special things when \( c = "\text{\textbackslash n}.\) \( \text{\textbackslash n}.\) \text{\{}n, \text{\}}n\); \textbf{goto done} at end 92) Used in section 91.

(Do the first pass of sorting 243) Used in section 239*.

(Emit the scrap for a module name if present 232) Used in section 230.

(Enter a new module name into the tree 67) Used in section 66.

(Enter a new name into the table at position \( p \) 62) Used in section 58.

(Error handling procedures 30, 31, 33*) Used in section 2*.

(Get a string 99) Used in section 95.

(Get an identifier 98) Used in section 95.

(Get control code and possible module name 100) Used in section 95.


(Go to \textbf{found} if \( c \) is a hexadecimal digit, otherwise set \textit{scanning.hex} \( \leftarrow \) false 96) Used in section 95.

(If end of name, \textbf{goto done} 104) Used in section 103.

(If semi-tracing, show the irreducible scraps 181) Used in section 180.

(If the current line starts with \texttt{\textbackslash y}, report any discrepancies and \textbf{return} 80) Used in section 79.

(If tracing, print an indication of where we are 182) Used in section 179.

(Invert the cross-reference list at \textit{cur.name}, making \textit{cur.xref} the head 255) Used in section 254.

(Local variables for initialization 16, 40, 56, 247) Used in section 2*.

(Look ahead for strongest line break, \textbf{goto reswitch} 212) Used in section 211.

(Make sure that there is room for at least four more scraps, six more tokens, and four more texts 187) Used in section 185.

(Make sure that there is room for at least seven more tokens, three more texts, and one more scrap 199) Used in section 198.

(Make sure the entries \texttt{cat\{pp . . (pp + 3)\}} are defined 176) Used in section 175.

(Match a production at \texttt{pp}, or increase \texttt{pp} if there is no match 149) Used in section 175.

(Move \texttt{buffer} and \texttt{limit} to \textit{change_buffer} and \textit{change_limit} 78) Used in sections 75 and 79.

(Output a control, look ahead in case of line breaks, possibly \textbf{goto reswitch} 211) Used in section 208.

(Output a \texttt{\textbackslash math} operator 210) Used in section 208.

(Output an module name 213) Used in section 208.

(Output all the module names 257) Used in section 239*.

(Output all the module numbers on the reference list \textit{cur.xref} 237) Used in section 236.

(Output an identifier 209) Used in section 208.

(Output index entries for the list at \textit{sort.ptr} 252) Used in section 250.

(Output the code for the beginning of a new module 221) Used in section 220.

(Output the code for the end of a module 238) Used in section 220.

(Output the cross-references at \textit{cur.name} 254) Used in section 252.

(Output the name at \textit{cur.name} 253) Used in section 252.

(Output the text of the module name 214) Used in section 213.

(Phase I: Read all the user’s text and store the cross references 109) Used in section 261*.

(Phase II: Read all the text again and translate it to \TeX{} form 218) Used in section 261*.

(Phase III: Output the cross-reference index 239*) Used in section 261*.

(Print error location based on input buffer 32) Used in section 31.

(Print error messages about unused or undefined module names 120) Used in section 109.

(Print statistics about memory usage 262) Used in section 33*.

(Print the job \textit{history} 263) Used in section 33*.

(Print token \texttt{r} in symbolic form 147) Used in section 146.

(Print warning message, break the line, \textbf{return} 128) Used in section 127.

(Process a format definition 116) Used in section 115.

(Put module name into \texttt{mod.text}[1 . . k] 103) Used in section 101.

(Read from \texttt{change_file} and maybe turn off \textit{changing} 84) Used in section 82.

(Read from \texttt{web_file} and maybe turn on \textit{changing} 83) Used in section 82.

(Rearrange the list pointed to by \textit{cur.xref} 235) Used in section 233.
(Reduce the scraps using the productions until no more rules apply 175)  Used in section 179.
(Scan a verbatim string 107)  Used in section 100.
(Scan the module name and make cur_module point to it 101)  Used in section 100.
(Scan to the next @ 106)  Used in section 100.
(Set initial values 10, 14, 17*, 18, 21*, 26*, 41, 43, 49, 54, 57, 94, 102, 124, 126, 145, 203, 245, 248, 259*)  Used in section 2*.
(Set variable c to the result of comparing the given name to name p 68)  Used in sections 66 and 69.
(Skip next character, give error if not '@' 215)  Used in section 214.
(Skip to the next nonblank line; return if end of file 76)  Used in section 75.
(Sort and output the index 250)  Used in section 239*.
(Skip over comment lines in the change file; return if end of file 77)  Used in section 75.
(Special control codes allowed only when debugging 88)  Used in section 87.
(Split the list at sort_ptr into further lists 251)  Used in section 250.
(Start a format definition 228)  Used in section 225.
(Start a macro definition 227)  Used in section 225.
(Store all the reserved words 64)  Used in section 261*.
(Store cross reference data for the current module 110)  Used in section 109.
(Store cross references in the definition part of a module 115)  Used in section 110.
(Store cross references in the Pascal part of a module 117)  Used in section 110.
(Store cross references in the \TeX{} part of a module 113)  Used in section 110.
(Tell about changed modules 241)  Used in section 239*.
(Translate a hexadecimal constant appearing in \TeX{} text 224)  Used in section 222.
(Translate an octal constant appearing in \TeX{} text 223)  Used in section 222.
(Translate the current module 220)  Used in section 218.
(Translate the definition part of the current module 225)  Used in section 220.
(Translate the Pascal part of the current module 230)  Used in section 220.
(Translate the \TeX{} part of the current module 222)  Used in section 220.
(Types in the outer block 11, 12*, 36, 38, 47, 52, 201)  Used in section 2*.