# The CTANGLE processor

(Version 4.7 [TeX Live])

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Data structures exclusive to CTANGLE</td>
<td>19</td>
</tr>
<tr>
<td>Tokens</td>
<td>26</td>
</tr>
<tr>
<td>Stacks for output</td>
<td>31</td>
</tr>
<tr>
<td>Producing the output</td>
<td>41</td>
</tr>
<tr>
<td>The big output switch</td>
<td>48</td>
</tr>
<tr>
<td>Introduction to the input phase</td>
<td>61</td>
</tr>
<tr>
<td>Inputting the next token</td>
<td>68</td>
</tr>
<tr>
<td>Scanning a macro definition</td>
<td>82</td>
</tr>
<tr>
<td>Scanning a section</td>
<td>90</td>
</tr>
<tr>
<td>Extensions to CWEB</td>
<td>104</td>
</tr>
<tr>
<td>Output file update</td>
<td>105</td>
</tr>
<tr>
<td>Print “version” information</td>
<td>116</td>
</tr>
<tr>
<td>Index</td>
<td>118</td>
</tr>
</tbody>
</table>

Copyright © 1987, 1990, 1993, 2000 Silvio Levy and Donald E. Knuth

Permission is granted to make and distribute verbatim copies of this document provided that the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this document under the conditions for verbatim copying, provided that the entire resulting derived work is given a different name and distributed under the terms of a permission notice identical to this one.
§1  CTANGLE (Version 4.7 [\TeX Live])  

1*  Introduction.  This is the CTANGLE program by Silvio Levy and Donald E. Knuth, based on TANGLE by Knuth. We are thankful to Nelson Beebe, Hans-Hermann Bode (to whom the C++ adaptation is due), Klaus Guntermann, Norman Ramsey, Tomas Rokicki, Joachim Schnitter, Joachim Schrod, Lee Wittenberg, and others who have contributed improvements.

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

```latex
\#define banner  "This is CTANGLE, Version 4.7" \end{verbatim}
```

2*  CTANGLE has a fairly straightforward outline. It operates in two phases: First it reads the source file, saving the C code in compressed form; then it shuffles and outputs the code.

Please read the documentation for COMMON, the set of routines common to CTANGLE and CWEAVE, before proceeding further.

```latex
int main(int ac, char **av)
{
    argc ← ac;argv ← av; program ← ctangle; (Set initial values 21)
    common_init();
    if (show_banner) cb_show_banner(); \end{verbatim}
    phase_one(); \end{verbatim}
    phase_two(); \end{verbatim}
    return wrap_up(); \end{verbatim}
```

3*  The next few sections contain stuff from the file "common.w" that must be included in both "ctangle.w" and "cweave.w". It appears in file "common.h", which is also included in "common.w" to propagate possible changes from this COMMON interface consistently.

First comes general stuff:

```latex
\langle Common code for CWEAVE and CTANGLE 3* \rangle≡
\begin{verbatim}
typedef uint8_t eight_bits;
typedef uint16_t sixteen_bits;
typedef enum {
    ctangle, cweave, ctwill
} cweb;
\end{verbatim}
```

See also sections 5*, 6*, 7*, 9*, 10*, 12*, 14*, 15*, and 116*.

This code is used in section 1*.  

You may have noticed that almost all "strings" in the CWB sources are placed in the context of the ‘\string’ macro. This is just a shortcut for the ‘\gettext’ function from the “GNU gettext utilities.” For systems that do not have this library installed, we wrap things for neutral behavior without internationalization. For backward compatibility with pre-ANSI compilers, we replace the “standard” header file ‘stdbool.h’ with the KPATHSEA interface ‘simpletypes.h’.

\#define _(s) gettext(s)
\langle Include files 4*\rangle
\#include <ctype.h>   \>
\#include <kpathsea/simpletypes.h> \>
\#include <stdbool.h> \>
\#include <stdint.h> \>
\#include <stdio.h> \>
\#include <stdlib.h> \>
\#include <string.h> \>
\#ifndef HAVE_GETTEXT
\#define HAVE_GETTEXT 0
\#endif
\#if HAVE_GETTEXT
\#include <libintl.h>
\#else
\#define gettext(a) a
\#endif

This code is used in section 1*.

5* Code related to the character set:

\#define and_and 04 \>
\#define lt_lt 020 \>
\#define gt_gt 021 \>
\#define plus_plus 013 \>
\#define minus_minus 01 \>
\#define minus_gt 031 \>
\#define non_eq 032 \>
\#define eq_eq 036 \>
\#define or_or 037 \>
\#define dot_dot_dot 016 \>
\#define colon_colon 06 \>
\#define period_ast 026 \>
\#define minus_gt_last 027 \>
\#define compress(c) if (loc++ <= limit) return c
\langle Common code for CWEAVE and CTANGLE 3*\rangle \equiv

extern char section_text[]; \>
extern char *section_text_end; \>
extern char *id_first; \>
extern char *id_loc;
6* Code related to input routines:

\[
\begin{align*}
\#define \text{xisalpha}(c) & \equiv (\text{isalpha}(\text{(int)}(c)) \land (\text{(eight_bits})(c) < \,^\circ 200)) \\
\#define \text{xisdigit}(c) & \equiv (\text{isdigit}(\text{(int)}(c)) \land (\text{(eight_bits})(c) < \,^\circ 200)) \\
\#define \text{xisspace}(c) & \equiv (\text{isspace}(\text{(int)}(c)) \land (\text{(eight_bits})(c) < \,^\circ 200)) \\
\#define \text{xislower}(c) & \equiv (\text{islower}(\text{(int)}(c)) \land (\text{(eight_bits})(c) < \,^\circ 200)) \\
\#define \text{xisupper}(c) & \equiv (\text{isupper}(\text{(int)}(c)) \land (\text{(eight_bits})(c) < \,^\circ 200)) \\
\#define \text{xisxdigit}(c) & \equiv (\text{isxdigit}(\text{(int)}(c)) \land (\text{(eight_bits})(c) < \,^\circ 200)) \\
\#define \text{ishalpha}(c) & \equiv (c \equiv \,^\circ '_-' \lor (c \equiv \,^\circ '$')) \land \text{non-alpha characters allowed in identifier} \\
\#define \text{ishigh}(c) & \equiv (\text{(eight_bits})(c) > \,^\circ 177)
\end{align*}
\]

(\text{Common code for CWEAVE and CTANGLE} 3* ) \equiv \equiv

\begin{align*}
\text{extern char } & \text{buffer}[\ ]; \triangleright \text{where each line of input goes} \\
\text{extern char } & \text{*buffer_end}; \triangleright \text{end of buffer} \\
\text{extern char } & \text{*loc}; \triangleright \text{points to the next character to be read from the buffer} \\
\text{extern char } & \text{*limit}; \triangleright \text{points to the last character in the buffer}
\end{align*}

7* Code related to file handling:

\begin{align*}
\text{format } & \text{line } x \triangleright \text{make line an unreserved word} \\
\#define \text{max\_include\_depth} & 10 \triangleright \text{maximum number of source files open simultaneously, not counting the change file} \\
\#define \text{max\_file\_name\_length} & 1024 \\
\#define \text{cur\_file\_file\_name}[\text{include\_depth}] & \triangleright \text{current file} \\
\#define \text{cur\_file\_name}[\text{include\_depth}] & \triangleright \text{current file name} \\
\#define \text{cur\_line}[\text{include\_depth}] & \triangleright \text{number of current line in current file} \\
\#define \text{web\_file\_buffer}[0] & \triangleright \text{main source file} \\
\#define \text{web\_file\_name}[\text{include\_depth}] & \triangleright \text{main source file name}
\end{align*}

(\text{Common code for CWEAVE and CTANGLE} 3* ) \equiv \equiv

\begin{align*}
\text{extern int } & \text{include\_depth}; \triangleright \text{current level of nesting} \\
\text{extern FILE } & \text{*file}[\ ]; \triangleright \text{stack of non-change files} \\
\text{extern FILE } & \text{*change\_file}; \triangleright \text{change file} \\
\text{extern char file\_name}[\text{max\_file\_name\_length}][\text{include\_depth}] & \triangleright \text{stack of non-change file names} \\
\text{extern char change\_file\_name}[\ ]; \triangleright \text{name of change file} \\
\text{extern char check\_file\_name}[\ ]; \triangleright \text{name of check\_file} \\
\text{extern int line}[\ ]; \triangleright \text{number of current line in the stacked files} \\
\text{extern int change\_line} & \triangleright \text{number of current line in change file} \\
\text{extern int change\_depth} & \triangleright \text{where \&0y originated during a change} \\
\text{extern boolean input\_has\_ended} & \triangleright \text{if there is no more input} \\
\text{extern boolean changing} & \triangleright \text{if the current line is from change\_file} \\
\text{extern boolean web\_file\_open} & \triangleright \text{if the web file is being read}
\end{align*}

8* (Predeclaration of procedures 8* ) \equiv

\begin{align*}
\text{extern boolean get\_line(\text{void})} & \triangleright \text{inputs the next line} \\
\text{extern void check\_complete(\text{void})} & \triangleright \text{checks that all changes were picked up} \\
\text{extern void reset\_input(\text{void})} & \triangleright \text{initialize to read the web file and change file}
\end{align*}


This code is used in section 1*.
9* Code related to section numbers:

\[\text{extern sixteen_bits section_count;} \quad \text{▷ the current section number} \]
\[\text{extern boolean changed_section[];} \quad \text{▷ is the section changed?} \]
\[\text{extern boolean change_pending;} \quad \text{▷ is a decision about change still unclear?} \]
\[\text{extern boolean print_where;} \quad \text{▷ tells CTANGLE to print line and file info} \]

10* Code related to identifier and section name storage:

\[\text{#define length(c) (size_t)((c + 1)-byte_start - (c)-byte_start)} \quad \text{▷ the length of a name} \]
\[\text{#define print_id(c) term_write((c)-byte_start, length(c))} \quad \text{▷ print identifier} \]
\[\text{#define llink link} \quad \text{▷ left link in binary search tree for section names} \]
\[\text{#define rlink dummy.Rlink} \quad \text{▷ right link in binary search tree for section names} \]
\[\text{#define root name_dir-link} \quad \text{▷ the root of the binary search tree for section names} \]

\[\text{typedef struct name_info} \{\]
\[\quad \text{char *byte_start;} \quad \text{▷ beginning of the name in byte_mem} \]
\[\quad \text{name_info *link;} \quad \text{▷ contains information about an identifier or section name} \]
\[\quad \text{union} \{\]
\[\quad \quad \text{name_info *Rlink;} \quad \text{▷ right link in binary search tree for section names} \]
\[\quad \quad \text{char Ilk;} \quad \text{▷ used by identifiers in CWEAVE only} \]
\[\quad \} \text{dummy}; \quad \text{▷ info corresponding to names} \]
\[\} \text{name_info}; \quad \text{▷ pointer into array of name_infos} \]

\[\text{typedef name_info *name_pointer;} \quad \text{▷ pointer into array of name_infos} \]
\[\text{typedef name_pointer *hash_pointer;} \quad \text{▷ pointer into array of name_infos} \]
\[\text{extern char byte_mem[];} \quad \text{▷ characters of names} \]
\[\text{extern char *byte_mem_end;} \quad \text{▷ end of byte_mem} \]
\[\text{extern char *byte_ptr;} \quad \text{▷ first unused position in byte_mem} \]
\[\text{extern name_info name_dir[];} \quad \text{▷ information about names} \]
\[\text{extern name_pointer name_dir_end;} \quad \text{▷ end of name_dir} \]
\[\text{extern name_pointer name_ptr;} \quad \text{▷ first unused position in name_dir} \]
\[\text{extern name_pointer hash[];} \quad \text{▷ heads of hash lists} \]
\[\text{extern hash_pointer hash_end;} \quad \text{▷ end of hash} \]
\[\text{extern hash_pointer h;} \quad \text{▷ index into hash-head array} \]

11* (Predeclaration of procedures 8*) \[\equiv \]
\[\text{extern boolean names_match(name_pointer, const char *, size_t, eight_bits);} \quad \text{▷ looks up a string in the identifier table} \]
\[\text{extern name_pointer id_lookup(const char *, const char *, eight_bits);} \quad \text{▷ finds section name} \]
\[\text{extern void init_node(name_pointer);} \quad \text{▷ initializes a node} \]
\[\text{extern void init_p(name_pointer, eight_bits);} \quad \text{▷ initializes a pointer} \]
\[\text{extern void print_prefix(name_pointer);} \quad \text{▷ prints the prefix} \]
\[\text{extern void print_section_name(name_pointer);} \quad \text{▷ prints the section name} \]
\[\text{extern void sprint_section_name(char *, name_pointer);} \quad \text{▷ prints the section name} \]
12* Code related to error handling:

```c
#define spotless 0  // history value for normal jobs
#define harmless_message 1  // history value when non-serious info was printed
#define error_message 2  // history value when an error was noted
#define fatal_message 3  // history value when we had to stop prematurely
#define mark_harmless if (history == spotless) history ← harmless_message
#define mark_error history ← error_message
#define confusion(s) fatal(_("This can't happen: ", s))
```

(Common code for CWEAVE and CTANGLE 3*) +≡

```c
extern int history;  // indicates how bad this run was
```

13* (Predeclaration of procedures 8*) +≡

```c
extern int wrap_up(void);  // indicate history and exit
extern void err_print(const char *);  // print error message and context
extern void fatal(const char *, const char *);  // issue error message and die
extern void overflow(const char *);  // succumb because a table has overflowed
```

14* Code related to command line arguments:

```c
#define show_banner_flags['b']  // should the banner line be printed?
#define show_progress_flags['p']  // should progress reports be printed?
#define show_happiness_flags['h']  // should lack of errors be announced?
#define show_stats_flags['s']  // should statistics be printed at end of run?
#define make_xrefs_flags['x']  // should cross references be output?
#define check_for_change_flags['c']  // check temporary output for changes
```

(Common code for CWEAVE and CTANGLE 3*) +≡

```c
extern int argv;  // copy of ac parameter to main
extern char **argv;  // copy of av parameter to main
extern char C_file_name[];  // name of C_file
extern char tex_file_name[];  // name of tex_file
extern char idx_file_name[];  // name of idx_file
extern char scn_file_name[];  // name of scn_file
extern boolean flags[];  // an option for each 7-bit code
extern const char *use_language;  // prefix to cwebmac.tex in \TeX output
```

15* Code related to output:

```c
#define update_terminal fflush(stdout)  // empty the terminal output buffer
#define new_line_putchar('\n')
#define term_write(a, b) fflush(stdout), fwrite(a, sizeof(char), b, stdout)
```

(Common code for CWEAVE and CTANGLE 3*) +≡

```c
extern FILE *C_file;  // where output of CTANGLE goes
extern FILE *tex_file;  // where output of CWEAVE goes
extern FILE *idx_file;  // where index from CWEAVE goes
extern FILE *scn_file;  // where list of sections from CWEAVE goes
extern FILE *active_file;  // currently active file for CWEAVE output
extern FILE *check_file;  // temporary output file
```
16* The procedure that gets everything rolling:

\begin{verbatim}
extern void common_init(void);
extern void print_stats(void);
extern void cb_show_banner(void);
\end{verbatim}

17* The following parameters are sufficient to handle \TeX{} (converted to \CWEB), so they should be sufficient for most applications of \CWEB.

\begin{verbatim}
#define buf_size 1000  // maximum length of input line, plus one
#define longest_name 10000  // file names, section names, and section texts shouldn't be longer than this
#define long_buf_size (buf_size + longest_name)  // for \CWEAVE
#define max_bytes 1000000  // the number of bytes in identifiers, index entries, and section names; must be less than $2^{24}$
#define max_names 10239  // number of identifiers, strings, section names; must be less than 10240
#define max_sections 4000  // greater than the total number of sections
\end{verbatim}

18* End of COMMON interface.

20* \begin{verbatim}
#define max_texts 10239  // number of replacement texts, must be less than 10240
#define max_toks 1000000  // number of bytes in compressed C code
\end{verbatim}

29* The following procedure is used to enter a two-byte value into \tok{} when a replacement text is being generated.

\begin{verbatim}
static void store_two_bytes(sixteen_bits x)
{
    if (tok_ptr + 2 > tok_mem_end) overflow(\texttt{"token");
    tok_ptr++ \leftarrow x \gg 8;  // store high byte
    tok_ptr++ \leftarrow x \& \texttt{\textasciicircum}377;  // store low byte
}\end{verbatim}
34* When the replacement text for name \( p \) is to be inserted into the output, the following subroutine is called to save the old level of output and get the new one going.

We assume that the C compiler can copy structures.

```c
static void push_level(name_pointer p)
{
    if (stack_ptr == stack_end) overflow(_("stack"));
    stack_ptr ← cur_state; stack_ptr ++;
    if (p \neq \Lambda) { /* p \equiv \Lambda means we are in output_defs */
        cur_name ← p; cur_repl ← (text_pointer) p-equiv; cur_byte ← cur_repl-tok_start;
        cur_end ← (cur_repl + 1)-tok_start; cur_section ← 0;
    }
}
```

40* The user may have forgotten to give any C text for a section name, or the C text may have been associated with a different name by mistake.

(Expand section \( a-\circ24000 \), goto restart 40*) \equiv

```c
{ a -= \circ24000; 
    if (((a + name_dir)-equiv \neq (void *) text_info) push_level(a + name_dir); 
    else if (a \neq 0) {
        fputs(_("\n!\ Not\ present: <"), stdout); print_section_name(a + name_dir); err_print(">");
    }
    goto restart;
}
```

This code is used in section 38.

47* (If it's not there, add cur_section_name to the output file stack, or complain we're out of room 47*) \equiv

```c
{ for (an_output_file ← cur_out_file; an_output_file \< end_output_files; an_output_file ++) 
    if (*an_output_file \equiv cur_section_name) break;
    if (an_output_file \equiv end_output_files) {
        if (cur_out_file \> output_files) *--cur_out_file ← cur_section_name;
        else overflow(_("output_files"));
    }
}
```

This code is used in section 77.
48* The big output switch. Here then is the routine that does the output.

```c
static void phase_two(void)
{
    phase ← 2; web_file_open ← false; cur_line ← 1;  \{Initialize the output stacks 33\}
\langle Output macro definitions if appropriate 51 \rangle
    if (text_info−text_link ≡ macro ∧ cur_out_file ≡ end_output_files) {
        fputs(\"\n!\nNo program text was specified\.", stdout); mark_harmless;
    } else {
        if (cur_out_file ≡ end_output_files) {
            if (show_progress) {
                printf(\"\nWriting the output file (\%s):\", C_file_name); update_terminal;
            }
        } else {
            if (show_progress) {
                fputs(\"\nWriting the output files:\", stdout); printf("\n(\%s)", C_file_name);
                update_terminal;
            }
            if (text_info−text_link ≡ macro) goto writeloop;
        }
    while (stack_ptr > stack) get_output();
    flush_buffer();
    writeloop: \{Write all the named output files 50*\}
        if (show_happiness) {
            if (show_progress) new_line;
            fputs(\"Done.\", stdout);
        }
}
```
To write the named output files, we proceed as for the unnamed section. The only subtlety is that we have to open each one.

\[ \text{Write all the named output files} \]

\[
\begin{align*}
\text{if} & \ (\text{check\_for\_change}) \ \{ \\
\quad & \text{fclose} (C\_file); \ C\_file \leftarrow \Lambda; \ (\text{Update the primary result when it has changed}) \ \{ \\
\quad & \text{fclose} (C\_file); \ C\_file \leftarrow \Lambda; \ (\text{Update the secondary results when they have changed}) \ \}\}
\end{align*}
\]

\[
\begin{align*}
\text{for} & \ (\text{an\_output\_file} \leftarrow \text{end\_output\_files}; \ \text{an\_output\_file} \succcurlyeq \text{cur\_out}; \ \{ \\
\quad & \text{an\_output\_file} \leftarrow \Lambda; \ \langle \text{Update the intermediate output file} \rangle \\
\quad & \text{fclose} (C\_file); \ C\_file \leftarrow \Lambda; \ (\text{Open the intermediate output file}) \\
\quad & \text{fclose} (C\_file); \ C\_file \leftarrow \Lambda; \ (\text{Update the primary result when it has changed}) \\
\quad & \text{fclose} (C\_file); \ C\_file \leftarrow \Lambda; \ (\text{Update the secondary results when they have changed}) \\
\quad & \text{fclose} (C\_file); \ C\_file \leftarrow \Lambda; \ (\text{Update the primary result when it has changed}) \\
\quad & \text{fclose} (C\_file); \ C\_file \leftarrow \Lambda; \ (\text{Update the secondary results when they have changed}) \\
\end{align*}
\]

\[
\begin{align*}
\text{if} & \ (\text{show\_progress}) \ \{ \\
\quad & \text{printf} ("\n(%s)", \text{output\_file\_name}); \ \text{update\_terminal}; \\
\quad & \text{cur\_name} \leftarrow \ast \text{an\_output\_file}; \ \text{cur\_repl} \leftarrow (\text{text\_pointer}) \\
\quad & \text{cur\_name-equiv}; \ \text{cur\_byte} \leftarrow \text{cur\_repl-tok\_start}; \ \text{cur\_end} \leftarrow (\text{cur\_repl} + 1)\text{-tok\_start}; \\
\quad & \text{flush\_buffer}(); \\
\quad & \text{fclose} (C\_file); \ C\_file \leftarrow \Lambda; \ (\text{Update the secondary results when they have changed}) \\
\end{align*}
\]

\[
\begin{align*}
\text{if} & \ (\text{check\_for\_change}) \ \{ \\
\quad & \text{strcpy} (\text{check\_file\_name}, \"\"); \ \quad \text{▷ We want to get rid of the temporary file} \ \} \\
\end{align*}
\]

This code is used in section 48.*
```c
#define C_printf(c, a) fprintf(C_file, c, a)
#define C_putchar(c) putc((int)(c), C_file)  // isn't C wonderfully consistent?

static void output_defs(void)
{
  sixteen_bits a;
  push_level(Λ);
  for (cur_text ← text_info + 1; cur_text < text_ptr; cur_text++)
    if (cur_text-text_link ≡ macro) {  // cur_text is the text for a macro
      cur_byte ← cur_text-tok_start; cur_end ← (cur_text + 1)-tok_start; C_printf("%s", "#define ∑");
      out_state ← normal; protect ← true;  // newlines should be preceded by '\\'
      while (cur_byte < cur_end) {
        a ← *cur_byte++;
        if (cur_byte ≡ cur_end ∧ a ≡ 'n') break;  // disregard a final newline
        if (out_state ≡ verbatim ∧ a ≠ string ∧ a ≠ constant ∧ a ≠ 'n') C_putchar(a);
          // a high-bit character can occur in a string
        else if (a < °200) out_char(a);  // one-byte token
        else {
          a ← (a - °200) * °400 + *cur_byte++;
          if (a < °24000) {  // °24000 ≡ (°250 - °200) * °400
            cur_val ← (int) a; out_char(identifier);
          }
        }
        else if (a < °50000) confusion(_("macro_defs_have_strange_char"));
        else {
          cur_val ← (int) a - °50000; cur_section ← (sixteen_bits) cur_val;
          out_char(section_number);
          // no other cases
        }
      }
    } else if (cur_byte ≡ cur_end ∧ a ≡ 'n') break;  // disregard a final newline
    if (out_state ≡ verbatim ∧ a ≠ string ∧ a ≠ constant ∧ a ≠ 'n') C_putchar(a);
      // a high-bit character can occur in a string
    else if (a < °200) out_char(a);  // one-byte token
    else {
      a ← (a - °200) * °400 + *cur_byte++;
      if (a < °24000) {  // °24000 ≡ (°250 - °200) * °400
        cur_val ← (int) a; out_char(identifier);
      }
    }
  }

  protect ← false; flush_buffer();
}

pop_level(false);
```
static boolean skip_comment(▷ skips over comments ◁
    boolean is_long_comment)
{
    char c;  ▷ current character ◁
    while (true) {
        if (loc > limit) {
            if (is_long_comment) {
                if (get_line()) return comment_continues ← true;
                else {
                    err_print("!Input ended in mid-comment"); return comment_continues ← false;
                }
            } else return comment_continues ← false;
        }
        c ← *(loc++);
        if (is_long_comment ∧ c ≡ '§' ∧ *loc ≡ '/') {
            loc++; return comment_continues ← false;
        }
        if (c ≡ '☑') {
            if (ccode[(eight_bits) *loc] ≡ new_section) {
                err_print("!Section name ended in mid-comment"); loc--;
                return comment_continues ← false;
            } else loc++;
        }
    }
}
C strings and character constants, delimited by double and single quotes, respectively, can contain newlines or instances of their own delimiters if they are protected by a backslash. We follow this convention, but do not allow the string to be longer than \texttt{longest\_name}.

\begin{verbatim}
( Get a string \texttt{74} ) \equiv \\
\{ char \ \textit{delim} \leftarrow (\texttt{char}) \textit{c}; \quad \triangleright \text{ what started the string } \triangleleft \\
\quad \textit{id\_first} \leftarrow \textit{section\_text} + 1; \ \textit{id\_loc} \leftarrow \textit{section\_text}; \ \ast\ast\ast \textit{id\_loc} \leftarrow \textit{delim}; \\
\quad \text{if} (\textit{delim} \equiv \texttt{L} \lor \textit{delim} \equiv \texttt{u} \lor \textit{delim} \equiv \texttt{U}) \{ \quad \triangleright \text{ wide character constant } \triangleleft \\
\quad \quad \text{if} (\textit{delim} \equiv \texttt{u} \land \ast\ast\ast \textit{loc} \equiv \texttt{8}) \ast\ast\ast \textit{id\_loc} \leftarrow \ast\ast\ast \textit{loc} + +; \\
\quad \quad \textit{delim} \leftarrow \ast\ast\ast \textit{loc} + +; \ \ast\ast\ast \textit{id\_loc} \leftarrow \textit{delim}; \\
\quad \}\text{ while } (\text{true}) \{ \\
\quad \text{if} (\textit{loc} \geq \textit{limit}) \{ \\
\quad \quad \text{if} (\ast(\textit{limit} - 1) \neq \texttt{\textbackslash\textbackslash}) \{ \\
\quad \quad \quad \text{err\_print}(\texttt{String didn't end}); \ \textit{loc} \leftarrow \textit{limit}; \ \text{break}; \\
\quad \quad \}\text{ if } (\textit{get\_line()} \equiv \text{false}) \{ \\
\quad \quad \quad \text{err\_print}(\texttt{Input ended in middle of string}); \ \textit{loc} \leftarrow \textit{buffer}; \ \text{break}; \\
\quad \quad \}\text{ else if } (\ast\ast\ast \textit{id\_loc} \leq \textit{section\_text\_end}) \ast\ast\ast \textit{id\_loc} \leftarrow \texttt{\textbackslash\textbackslash}; \quad \triangleright \text{ will print as } \texttt{\textbackslash\textbackslash\textbackslash}; \triangleleft \\
\quad \}\text{ if } ((\textit{c} \leftarrow (\texttt{eight\_bits}) \ast\ast\ast \textit{loc}) \equiv \textit{delim}) \{ \\
\quad \quad \text{if } (\ast\ast\ast \textit{id\_loc} \leq \textit{section\_text\_end}) \ast\ast\ast \textit{id\_loc} \leftarrow (\texttt{char}) \textit{c}; \\
\quad \}\text{ break; } \\
\quad \text{if } (\textit{c} \equiv \texttt{\textbackslash\textbackslash}) \{ \\
\quad \quad \text{if } (\textit{loc} \geq \textit{limit}) \text{ continue}; \\
\quad \quad \text{if } (\ast\ast\ast \textit{id\_loc} \leq \textit{section\_text\_end}) \ast\ast\ast \textit{id\_loc} \leftarrow \texttt{\textbackslash\textbackslash}; \\
\quad \quad \textit{c} \leftarrow (\texttt{eight\_bits}) \ast\ast\ast \textit{loc} + +; \\
\quad \}\text{ if } (\ast\ast\ast \textit{id\_loc} \leq \textit{section\_text\_end}) \ast\ast\ast \textit{id\_loc} \leftarrow (\texttt{char}) \textit{c}; \\
\}\text{ if } (\textit{id\_loc} \geq \textit{section\_text\_end}) \{ \\
\quad \text{fputs}(\texttt{String too long:}, \textit{stdout}); \ \text{term\_write}(\textit{section\_text} + 1, 25); \ \text{err\_print}(\ldots); \\
\quad \textit{id\_loc} + +; \ \text{return } \texttt{string}; \\
\}\text{This code is used in section 69.}
\end{verbatim}
75* After an @ sign has been scanned, the next character tells us whether there is more work to do.

\[
\text{(Get control code and possible section name 75*) } \equiv \\
\text{ switch } (c \leftarrow c \text{code} [(\text{eight_bits}) \ast \text{loc}++] ) \{ \\
\text{ case ignore: continue; } \\
\text{ case translit\_code: err\_print(\text{!\_\_Use\_\_@\_\_in\_\_limbo\_\_only}); continue; } \\
\text{ case control\_text: } \\
\quad \text{ while } ((c \leftarrow \text{skip\_ahead()} ) \equiv '\@') ; \quad \triangleright \text{ only @ and @> are expected } < \\
\quad \text{ if } (*\text{loc} - 1 ) \neq '\@') \quad \text{err\_print(\text{!\_\_Double\_\_@\_\_should\_\_be\_\_used\_\_in\_\_control\_\_text}); } \\
\quad \text{ continue; } \\
\text{ case section\_name: cur\_section\_name\_char } \leftarrow * (\text{loc} - 1 ) ; \quad \langle \text{Scan the section name and make cur\_section\_name point to it 77} \rangle \\
\text{ case string: } \langle \text{Scan a verbatim string 81*} \rangle \\
\text{ case ord: } \langle \text{Scan an ASCII constant 76*} \rangle \\
\text{ default: return c; } \\
\} \\
\]
This code is cited in section 92.

This code is used in section 69.

76* After scanning a valid ASCII constant that follows @', this code plows ahead until it finds the next single quote. (Special care is taken if the quote is part of the constant.) Anything after a valid ASCII constant is ignored; thus, @'\nopq' gives the same result as @'\n'.

\[
\text{(Scan an ASCII constant 76*) } \equiv \\
\text{ id\_first } \leftarrow \text{loc}; \\
\text{ if } (*\text{loc} \equiv '\\backslash') \\
\quad \text{ if } (*\text{loc} + 1 ) \equiv '\\backslash') \text{ loc++; } \\
\text{ while } (*\text{loc} \neq '\\backslash') \{ \\
\quad \text{ if } (*\text{loc} \equiv '\@') \{ \\
\quad \quad \text{ if } (*\text{loc} + 1 ) \neq '\@') \text{ err\_print(\text{!\_\_Double\_\_@\_\_should\_\_be\_\_used\_\_in\_\_ASCII\_\_constant}); } \\
\quad \quad \text{ else loc++;} \\
\quad \} \\
\text{ loc++; } \\
\text{ if } (\text{loc} > \text{limit}) \{ \\
\quad \text{err\_print(\text{!\_\_String\_\_didn’t\_\_end}); loc } \leftarrow \text{limit} - 1 ; \text{ break;} \\
\} \\
\} \\
\text{loc++; return ord; } \\
\]
This code is used in section 75*.
\[79* \quad \text{(Put section name into section_text 79*)} \equiv \]

\[\text{while (true) \{}
\text{\quad \text{if (loc > limit \&\& get_line() \equiv false) \{}
\text{\quad \quad \text{err_print("\!\text{Input ended in \text{section_name}"}); loc \leftarrow buffer + 1; \ break;}
\text{\quad \}}
\text{\quad c \leftarrow (eight_bits) *loc; \ (If end of name or erroneous nesting, break 80*)}
\text{\quad loc++;}
\text{\quad if (k < section_text_end) k++;}
\text{\quad if (xisspace(c)) \{
\text{\quad \quad c \leftarrow (eight_bits) ' ';
\text{\quad \quad if (* (k - 1) \equiv ' ') k--;}
\text{\quad \}}}
\text{\quad *k \leftarrow (char) c;
\text{\}}}
\text{\quad if (k \geq section_text_end) \{
\text{\quad \quad fputs(_("\!\text{Section name too long:"), stdout); term_write(section_text + 1, 25); printf("..."); mark_harmless;}
\text{\quad \}}
\text{\quad if (*k \equiv ' ' \&\& k > section_text) k--;
\text{\}}\]

This code is used in section 77.

\[80* \quad \text{(If end of name or erroneous nesting, break 80*)} \equiv \]

\[\text{\quad if (c \equiv ' @' \{}
\text{\quad \quad c \leftarrow (eight_bits) *(loc + 1);
\text{\quad \quad if (c \equiv '>') \{
\text{\quad \quad \quad loc += 2; \ break;}
\text{\quad \quad \}}}
\text{\quad \quad if (ccode[(eight_bits) c] \equiv new_section) \{
\text{\quad \quad \quad err_print(_("\!\text{Section name didn't end}")); break;}
\text{\quad \quad \}}}
\text{\quad \quad if (ccode[(eight_bits) c] \equiv section_name) \{
\text{\quad \quad \quad err_print(_("\!\text{Nesting of section names not allowed}")); break;}
\text{\quad \quad \}}}
\text{\quad \quad * (++k) \leftarrow ' @'; \ loc++; \quad \text{\textit{\textgreater now c \equiv \textit{loc} again \textless}}
\text{\quad \}}\]

This code is used in section 79*.

\[81* \quad \text{At the present point in the program we have *(loc - 1) \equiv string; we set id_first to the beginning of the string itself, and id_loc to its ending-plus-one location in the buffer. We also set loc to the position just after the ending delimiter.}
\text{\quad \text{(Scan a verbatim string 81*)} \equiv \]
\text{\quad \quad id_first \leftarrow loc + ; \ *(limit + 1) \leftarrow ' @'; \ *(limit + 2) \leftarrow '>';}
\text{\quad \quad while (*loc \neq ' @' \&\& * (loc + 1) \neq '>') loc ++;}
\text{\quad \quad if (loc \geq limit) err_print(_("\!\text{Verbatim string didn't end}"));}
\text{\quad \quad id_loc \leftarrow loc; \ loc += 2; \ \textbf{return string;}
\text{\quad \}}\]

This code is used in section 75*. 
Scanning a macro definition. The rules for generating the replacement texts corresponding to macros and C texts of a section are almost identical; the only differences are that

a) Section names are not allowed in macros; in fact, the appearance of a section name terminates such macros and denotes the name of the current section.

b) The symbols @d and @f and @c are not allowed after section names, while they terminate macro definitions.

c) Spaces are inserted after right parentheses in macros, because the ANSI C preprocessor sometimes requires it.

Therefore there is a single procedure scan_repl whose parameter t specifies either macro or section name. After scan_repl has acted, cur_text will point to the replacement text just generated, and next_control will contain the control code that terminated the activity.

```c
#define app_repl(c)
{
    if (tok_ptr == tok_mem_end) overflow(\"token\"); 
    else *(tok_ptr++) = (eight_bits)c;
}

⟨Private variables 20*⟩ += static text_pointer cur_text;  ▷ replacement text formed by scan_repl <
static eight_bits next_control;

83* static void scan_repl(  ▷ creates a replacement text <
eight_bits t)
{
    sixteen_bits a;  ▷ the current token <
    if (t == section_name) { Insert the line number into tok_mem 85
        while (true)
            switch (a ← get_next()) {
                ⟨In cases that a is a non-char token (identifier, section_name, etc.), either process it and change a
                to a byte that should be stored, or continue if a should be ignored, or goto done if a signals
                the end of this replacement text 86*⟩
                case ')': app_repl(a);
                    if (t == macro) app_repl('\u');
                    break;
                default: app_repl(a);  ▷ store a in tok_mem <
            }
    done: next_control ← (eight_bits)a;
        if (text_ptr > text_info_end) overflow(\"text\");
        cur_text ← text_ptr; (++text_ptr)-tok_start ← tok_ptr;
    }
```
In cases that \texttt{a} is a non-\texttt{char} token (\texttt{identifier}, \texttt{section\_name}, etc.), either process it and change \texttt{a} to a byte that should be stored, or \texttt{continue} if \texttt{a} should be ignored, or \texttt{goto done} if \texttt{a} signals the end of this replacement text. 

\begin{verbatim}
86* case identifier: store_id(a);
  if (*buffer == '#' \&\& ((id_loc - id_first) \equiv 5 \&\& strncmp("endif", id_first, 5) \equiv 0) \lor
    (id_loc - id_first) \equiv 4 \&\& strncmp("else", id_first, 4) \equiv 0) \lor
    (id_loc - id_first) \equiv 4 \&\& strncmp("elif", id_first, 4) \equiv 0)) \triangleright Avoid preprocessor calamities \langle
    print_where \leftarrow true;
  break;

86* case section\_name:
  if (t \neq section\_name) goto done;
  else {
    a \leftarrow cur\_section\_name - name\_dir; app\_repl((a/°400) + °250); app\_repl(a \%°400);
    \langle Insert the line number into tok\_mem \rangle
  }
  break;

87* case output\_defs\_code:
  if (t \neq section\_name) err\_print(\texttt{(!\_\_\_Misplaced\_\_\_h)});
  else {
    output\_defs\_seen \leftarrow true; a \leftarrow output\_defs\_flag; app\_repl((a/°400) + °200); app\_repl(a \%°400);
    \langle Insert the line number into tok\_mem \rangle
  }
  break;

88* case constant: case string: \langle Copy a string or verbatim construction or numerical constant \rangle
  break;

89* case ord: \langle Copy an ASCII constant \rangle
  break;

90* case definition: case format\_code: case begin\_C:
  if (t \neq section\_name) goto done;
  else {
    err\_print(\texttt{(!\_\_\_Missing\_\_\_ before\_\_\_named\_\_\_section)});
    continue;
  }

91* case new\_section: goto done;
\end{verbatim}

This code is used in section 86*. 

\begin{verbatim}
87* \langle Was an ‘@’ missed here? \rangle
{
  char *try\_loc \leftarrow loc;
  while (*try\_loc == '_' \&\& try\_loc < limit) try\_loc++;
  if (*try\_loc == '+' \&\& try\_loc < limit) try\_loc++;
  while (*try\_loc == '_' \&\& try\_loc < limit) try\_loc++;
  if (*try\_loc == '=') err\_print(\texttt{(!\_\_\_Missing\_\_\_ before\_\_\_named\_\_\_section)});
    \triangleright user who isn’t defining a section should put newline after the name, as explained in the manual
}
This code is used in section 86*.
\end{verbatim}
By default, CTANGLE purges single-quote characters from C++-style literals, e.g., 1’000’000, so that you can use this notation also in C code. The \( \ast k \) switch will ‘keep’ the single quotes in the output.

```c
#define keep_digit_separators flags ['k']  
⟨ Copy a string or verbatim construction or numerical constant \( \ast 88 \ast \) ⟩ ≡  
app repl (a);  ▷ string or constant < 
while (id_first < id_loc) {  ▷ simplify @@ pairs < 
  if (*id_first ≡ '@') {  
    if (*id_first + 1 ≡ '@') id_first++;  
    else err_print(\"Double @ should be used in string\"));  
  }  
  else if (a ≡ constant ∧ *id_first ≡ '\'' ∧ ¬keep_digit_separators) id_first++;  
  app repl (*id_first ++);  
}  
app repl (a);  
```

This code is used in section \( \ast 86 \ast \).
This section should be rewritten on machines that don’t use ASCII code internally.

(Copy an ASCII constant $89^*$) \(\equiv\)

```c
int c ← (int)((eight_bits) * id_first);
if (c ≡ \'\\' \{ c ← (int)((eight_bits) * ++id_first); if (c ≥ \'0' ∧ c ≤ \'7\) \{ c ← \'0\;
    if (*id_first + 1) ≥ \'0' ∧ (*id_first + 1) ≤ \'7\) \{ c ← 8 * c + * (++id_first) \-'0\';
    if (*id_first + 1) ≥ \'0' ∧ (*id_first + 1) ≤ \'7\ ∧ c < 32) c ← 8 * c + * (++id_first) \-'0\';
```

```c
} else
    switch (c) { case \'t\': c ← \'t\'; break;
    case \'n\': c ← \'n\'; break;
    case \'b\': c ← \'b\'; break;
    case \'f\': c ← \'f\'; break;
    case \'v\': c ← \'v\'; break;
    case \'r\': c ← \'r\'; break;
    case \'a\': c ← \'7\'; break;
    case \?'': c ← \?\'; break;
    case \'x\':
        if (xisdigit(*id_first + 1)) c ← (int)* (++id_first) \-'0\';
        else if (xisxdigit(*id_first + 1)) \{ ++id_first; c ← toupper((int)*id_first) \-'A\' + 10;
```
93* (Scan a definition 93*) ≡
while ((next_control ← get_next()) ≡ \n) ; ▷ allow newline before definition ≡
if (next_control ≠ identifier) {
  err_print(’!_Definition flushed, must start with identifier’); continue;
}
store_id(a); ▷ append the lhs ≡
if (*loc ≠ ’‘) { ▷ identifier must be separated from replacement text ≡
  app_repl(string); app_repl(’\l’); app_repl(string);
}
scan_repl(macro); cur_text→text_link ← macro;

This code is used in section 90.

100* Only a small subset of the control codes is legal in limbo, so limbo processing is straightforward.

static void skip_limbo(void)
{
  while (true) {
    if ((loc > limit ∧ get_line()) ≡ false) return;
    *limit + 1 ← ’@’;
    while (*loc ≠ ’@’) loc ++;
    if (loc ++ ≤ limit) {
      char c ← *loc ++;
      switch (ccode[eight_bits] c) {
        case new_section: return;
        case translit_code:  (Read in transliteration of a character 102*)
          break;
        case format_code:  case ’@’: break;
        case control_text:
          if (c ≡ ’q’ ∨ c ≡ ’Q’) {
            while ((c ← (char) skip_ahead()) ≡ ’@’);
            if (*loc − 1 ≠ ’>’) err_print(’!_Double@ should be used in control.text’);
            break;
          }
          /* otherwise fall through */
        default: err_print(’!_Double@ should be used in limbo’);
      }
    }
  }
102* (Read in transliteration of a character 102*) \(\equiv\)

\[
\text{while } (\text{isspace}(*loc) \land loc < \text{limit}) \text{ loc}++; \\
\text{loc } += 3;
\]

\[
\text{if } (loc > \text{limit} \lor \neg \text{xisdigit}(*(loc - 3)) \lor \neg \text{xisdigit}(*(loc - 2)) \\
\lor *(loc - 3) \geq \text{'}0' \land *(loc - 3) \leq \text{'}7' \lor \neg \text{isspace}(*(loc - 1))
\]

err_print(\("!_\text{Improper}_\text{hex}_\text{number}_\text{following}_@l"));

else {
    \text{unsigned int } i;
    \text{char } *beg;
    \text{sscanf}((loc - 3), \"%x\", &i);
    \text{while } (\text{isspace}(*loc) \land loc < \text{limit}) \text{ loc}++; \\
    \text{beg } \leftarrow \text{loc};
    \text{while } (loc < \text{limit} \land (\text{xisalpha}(*loc) \lor \text{xisdigit}(*loc) \lor \text{loc } \equiv \text{'}_') \land \text{loc}) \text{ loc}++;
    \text{if } (loc - \text{beg } \geq \text{translit_length}) \text{ err_print}(\("!_\text{Replacement}_\text{string}_\text{in}_@l_\text{too}_\text{long}"));
    \text{else } \{ \\
    \text{strncpy}((i - 200), \text{beg}, (\text{size_t})(loc - \text{beg})); \text{translit}[i - 200][\text{loc} - \text{beg}] \leftarrow \text{'}0';
    \}
\}

This code is used in section 100*.

103* Because on some systems the difference between two pointers is a \texttt{ptrdiff_t} but not an \texttt{int}, we use \texttt{\%td} to print these quantities.

\textbf{void} print_stats(\textbf{void})

\{
    \text{puts}(\("\text{\n\ Memory usage statistics:}\"));
    \text{printf}(\("\text{\n\ %td\ names (out of %ld)}\"), (\texttt{ptrdiff_t})(\text{name}\_ptr - \text{name}\_dir), (\texttt{long}) \text{max}\_names);
    \text{printf}(\("\text{\n\ %td\ replacement texts (out of %ld)}\"), (\texttt{ptrdiff_t})(\text{text}\_ptr - \text{text}\_info), (\texttt{long}) \text{max}\_texts);
    \text{printf}(\("\text{\n\ %td\ bytes (out of %ld)}\"), (\texttt{ptrdiff_t})(\text{byte}\_ptr - \text{byte}\_mem), (\texttt{long}) \text{max}\_bytes);
    \text{printf}(\("\text{\n\ %td\ tokens (out of %ld)}\"), (\texttt{ptrdiff_t})(\text{tok}\_ptr - \text{tok}\_mem), (\texttt{long}) \text{max}\_toks);
\}
104* Extensions to CWEB. The following sections introduce new or improved features that have been created by numerous contributors over the course of a quarter century.

Care has been taken to keep the original section numbering intact, so this new material should nicely integrate with the original “104. Index.”
**Output file update.** Most C projects are controlled by a Makefile that automatically takes care of the temporal dependencies between the different source modules. It may be convenient that CWEB doesn’t create new output for all existing files, when there are only changes to some of them. Thus the make process will only recompile those modules where necessary. You can activate this feature with the ‘+c’ command-line option. The idea and basic implementation of this mechanism can be found in the program NUWEB by Preston Briggs, to whom credit is due.

This code is used in section 50*.

```c
if ((C_file ← fopen(output_file_name, "a")) ≡ Λ)
    fatal(_."Cannot open output_file"), output_file_name);
else fclose(C_file);
▷ Test accessibility ◀
if ((C_file ← fopen(check_file_name, "wb")) ≡ Λ)
    fatal(_."Cannot open output_file"), check_file_name);

This code is used in section 50*.
```

This code is used in section 50*.

```c
if ((C_file ← fopen(C_file_name, "r")) ≠ Λ) {
    (Set up the comparison of temporary output 107*)
    (Create the primary output depending on the comparison 109*)
}
else rename(check_file_name, C_file_name);  ▷ This was the first run ◀
```

This code is used in section 50*.

```c
boolean comparison ← false;
if ((check_file ← fopen(check_file_name, "r")) ≡ Λ)
    fatal(_."Cannot open output_file"), check_file_name);
(Compare the temporary output to the previous output 108*)
fclose(C_file); C_file ← Λ; fclose(check_file); check_file ← Λ;
```

This code is used in sections 106* and 110*.

This code is used in section 50*.

108* We hope that this runs fast on most systems.

(Compare the temporary output to the previous output 108*)

```c
do {
    char x[BUFSIZ], y[BUFSIZ];
    int x.size ← fread(x, sizeof(char), BUFSIZ, C_file);
    int y.size ← fread(y, sizeof(char), BUFSIZ, check_file);
    comparison ← (x.size ≡ y.size) ∧ ¬memcmp(x, y, x.size);
} while (comparison ∧ ¬feof(C_file) ∧ ¬feof(check_file));
```

This code is used in section 107*.

109* Note the superfluous call to remove before rename. We’re using it to get around a bug in some implementations of rename.

(Compare the primary output to the previous output 108*)

```c
if (comparison) remove(check_file_name);  ▷ The output remains untouched ◀
else {
    remove(C_file_name); rename(check_file_name, C_file_name);
}
```

This code is used in section 106*.
The author of a CWEB program may want to write the secondary output instead of to a file (in \@(...@\>) to /dev/null or /dev/stdout or /dev/stderr. We must take care of the temporary output already written to a file and finally get rid of that file.

\begin{verbatim}
if (0 \equiv strcmp("/dev/stdout", output_file_name))
  Redirect temporary output to /dev/stdout
else if (0 \equiv strcmp("/dev/stderr", output_file_name))
  Redirect temporary output to /dev/stderr
else if (0 \equiv strcmp("/dev/null", output_file_name))
  Redirect temporary output to /dev/null
else {
  \> Hopefully a regular output file
  if ((C_file \leftarrow fopen(output_file_name,"r")) \neq \Lambda)
    \> Set up the comparison of temporary output
    \> Create the secondary output depending on the comparison
  
else rename(check_file_name, output_file_name); \> This was the first run
}
\end{verbatim}

This code is used in section 50*.

Again, we use a call to remove before rename.

\begin{verbatim}
if (comparison) remove(check_file_name); \> The output remains untouched
else {

remove(output_file_name); rename(check_file_name, output_file_name);
}
\end{verbatim}

This code is used in sections 110*, 112*, 113*, and 114*.

Copy secondary output to stdout.

\begin{verbatim}
do {
  in_size \leftarrow fread(in_buf, sizeof(char), BUFSIZE, check_file);
  in_buf[in_size] \leftarrow '\0';
  fprintf(stdout, "%s", in_buf);
} while (\neg feof(check_file));
fclose(check_file); check_file \leftarrow \Lambda; \> Create the secondary output depending on the comparison
\end{verbatim}

This code is used in section 110*.

Copy secondary output to stderr.

\begin{verbatim}
do {
  in_size \leftarrow fread(in_buf, sizeof(char), BUFSIZE, check_file);
  in_buf[in_size] \leftarrow '\0';
  fprintf(stderr, "%s", in_buf);
} while (\neg feof(check_file));
fclose(check_file); check_file \leftarrow \Lambda; \> Create the secondary output depending on the comparison
\end{verbatim}

This code is used in section 110*.
No copying necessary, just remove the temporary output file.

(Redirect temporary output to /dev/null 114*) \equiv

\{ 
  \textit{boolean} \textit{comparison} \leftarrow \textit{true}; 
  \langle \textit{Create the secondary output depending on the comparison 111*} \rangle 
\}

This code is used in section 110*.

\quad (\textit{Setup system redirection 115*}) \equiv

\textit{char} \textit{in}._{buf}[\text{BUFSIZ} + 1];
\textit{int} \textit{in}._{size};
\textit{boolean} \textit{comparison} \leftarrow \textit{true};
\textbf{if} (((\textit{check}._{file} \leftarrow \textit{fopen}((\textit{check}._{file}._{name}, "r")) \equiv \Lambda)
\textbf{fatal}(\langle !.\textit{Cannot}._{open}._{output}._{file}\rangle, \textit{check}._{file}._{name});

This code is used in sections 112* and 113*. 
Print “version” information. Don’t do this at home, kids! Push our local macro to the variable in COMMON for printing the banner and the versionstring from there.

#define max_banner 50
⟨Common code for CwEAVE and CTANGLE 3⟩ +≡
  extern char cb_banner[];

⟨Set initial values 21⟩ +≡
  strncpy(cb_banner, banner, max_banner - 1);
118* Index. Here is a cross-reference table for CTANGLE. All sections 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 section names are not indexed. Underlined entries correspond to where the identifier was declared. Error messages and a few other things like "ASCII code dependencies" are indexed here too.

The following sections were changed by the change file: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 20, 29, 34, 40, 47, 48, 50, 54, 67, 74, 75, 76, 79, 80, 81, 82, 83, 86, 87, 88, 89, 93, 100, 102, 103, 104, 105, 106, 107, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118.

@d, @f and @c are ignored in C text: 86*.
: 4*
ac: 38, 54* 60, 83*, 85, 90.
active_file: 15*
an_output_file: 45, 47*, 50*
and_and: 5*, 56, 71.
app_repl: 82*, 83*, 85, 86*, 88*, 89*, 93*
argc: 14*
argv: 2*, 14*.
ASCII code dependencies: 5*, 28, 89*.
av: 2* 14*.
banner: 1* 116*, 117*.
beg: 102*.
begin_C: 62, 63, 86*, 90.
buf_size: 17*.
buffer: 6* 69, 74*, 79*, 86*.
buffer_end: 6*.
BUFSIZ: 108*, 112*, 113*, 115*.
byte_field: 31, 32.
byte_mem: 10*, 19, 55, 103*.
byte_mem_end: 10*.
byte_ptr: 10* 103*.
byte_start: 10* 24, 31, 59, 60.
c: 63, 64, 67*, 69*, 80*, 100*.
C_printf: 54*, 59, 60.
C_putchar: 38, 43, 54*, 55, 56, 59, 60.
Cannot open output file: 50*, 105*, 107*, 115*.
check_file: 116* 117*.
check_file: 7* 85.
check_file_name: 7* 85.
change_depth: 7* 85.
change_line: 7* 85.
check_complete: 8* 98.
check_for_change: 14* 50*.
colon_colon: 5* 56, 71.
comment_continues: 14*, 56*, 69.
common_init: 2*, 16*.
compress: 5* 71.
confusion: 12*, 54*.
constant: 28, 38, 54*, 55, 73, 86*, 88*, 89*.
control_text: 62*, 63, 75*, 100*.
ctangle: 2*, 3*.
cweb: 3*.
cur_char: 55, 60.
cur_end: 31, 32, 33, 34*, 36, 38, 50*, 54*.
cur_file: 7*.
cur_file_name: 7* 85.
cur_line: 7* 43, 48*, 50*, 85.
cur_name: 31, 32, 33, 34*, 50*.
cur_out_file: 45, 46, 47*, 48*, 50*.
cur_repl: 31, 32, 33, 34*, 36, 50*.
cur_section: 31, 32, 33, 34*, 38, 54*.
cur_section_name: 47*, 68*, 77*, 86*, 90.
cur_section_name_char: 45*, 75*, 77.
cur_text: 54*, 82*, 83*, 93*, 95, 97.
cur_val: 37, 38, 54*, 59, 60.
cweave: 3*.
cweb: 3*.
definition: 62, 63, 86*, 90, 92.
Definition flushed...: 93*.
delim: 74*.
done: 83*, 86*.
dot_dot_dot: 5*, 56, 71.
Double @ should be used...: 75*, 76*, 88*, 100*.
dummy: 10*.
end_field: 31, 32.
end_output_files: 45, 46, 47*, 48*, 50*.
eq_eq: 5*, 56, 71, 94.
equiv_or_xref: 10* 22.
err_print: 13* 40* 41* 67* 74* 75* 76* 79* 80* 81* 86* 87* 88* 89* 93* 100* 102*
error_message: 12*
exit: 4*
false: 4* 36* 48* 52* 54* 64* 66* 67* 69* 73* 74* 77* 79* 90* 100* 107*
fatal: 12* 13* 50* 105* 107* 115*
fatal_message: 12*close: 50* 105* 107* 112* 113*eito: 108* 112* 113*write: 15*
flush: 7*
file: 7*
filename: 7*
first: 24.
flag: 36.
flags: 14* 88*
flush_buffer: 43* 44* 48* 50* 54* 55.
open: 50* 105* 106* 107* 110* 115*
form_code: 62* 63* 86* 100*
found: 73.
printf: 54* 112* 113*
puts: 40* 48* 74* 79*
fread: 108* 112* 113*
fwrite: 15*
get_line: 8* 64* 67* 69* 74* 79* 100*
get_next: 66* 69* 70* 83* 92* 93* 94.
get_output: 37* 38* 39* 41* 48* 50*
getenv: 4*
getext: 4*
gt_eq: 5* 56* 71.
gt_gt: 5* 56* 71.
h: 10*
allo_message: 12*
hash: 10*
hash_end: 10*
hash_pointer: 10*
HAVE_GETTEXT: 4*
hex_flag: 73.
high-bit character handling: 6* 38* 54* 59.
history: 12* 13*
i: 58* 102*
id_first: 5* 72* 73* 74* 76* 81* 85* 86* 88* 89*
id_loc: 5* 72* 73* 74* 81* 85* 86* 88*
id_lookup: 11* 85.
identifier: 37* 38* 54* 59* 72* 86* 93*
idx_file: 14* 15*
idx_file_name: 14*
ignore: 62* 63* 64* 75* 90.
Ilk: 10*
Improper hex number...: 102*
in_buf: 112* 113* 115*
in_size: 112* 113* 115*
include_depth: 7* 85.
init_node: 11* 23* 25.
init_p: 11* 25.
Input ended in mid-comment: 67*
Input ended in middle of string: 74*
Input ended in section name: 79*
input_has Ended: 77* 98.
is_long_comment: 66* 67*
isdigit: 4* 6* 69* 72.
ishigh: 6* 69* 72.
islower: 6*
isspace: 6*
isupper: 6*
isxalpha: 6* 69* 72.
isxdigit: 6*
j: 55.
join: 28* 55* 63.
k: 55* 77.
keep_digit_separators: 88*
l: 24.
last_unnamed: 26* 27* 97.
length: 10* 24.
limit: 56* 64* 67* 69* 74* 76* 79* 81* 87* 100* 102*
line: 7*
#line: 60.
link: 10*
link: 10*
loc: 5* 6* 64* 67* 69* 71* 72* 73* 74* 75* 76* 79* 80* 81* 87* 90* 92* 93* 100* 102*
long_buf_size: 17*
longest_name: 17* 45* 74*
lterror: 5* 56* 71.
l DriverManager: 5* 56* 71.
macro: 26* 27* 48* 54* 82* 83* 93*
main: 2* 14*
make_xrefs: 14*
mark_error: 12*
mark_44_message: 12* 48* 79*
max_banner: 116* 117*
max_bytes: 17* 103*
max_file_name_length: 7*
max_files: 45* 46.
max_includes: 7*
max_names: 17* 103*
max_sections: 17*
max_text: 20* 26* 103*
max_toks: 20* 103*
memcmp: 108*
minus_gt: 5* 56* 71.
minus_gt_ast: 5* 56* 71.
store_two_bytes: \texttt{29, 30, 85, 96}.
strcp: \texttt{4, 110}.
strcpy: \texttt{50}.
string: \texttt{28, 38, 54, 55, 63, 74, 75, 81, 86, 88, 93}.
String didn’t end: \texttt{74, 76}.
String too long: \texttt{74}.
strlen: \texttt{4, 85}.
strncpy: \texttt{24, 77, 86}.
system dependencies: \texttt{34}.
t: \texttt{24, 25, 83}.
term_write: \texttt{10, 15, 74, 79}.
tex_file: \texttt{14, 15}.
tex_file_name: \texttt{14}.
text: \texttt{10, 20}.
text_info: \texttt{19, 20, 21, 22, 25, 26, 27, 33, 36, 40, 48, 54, 97, 103}.
text_info_end: \texttt{20, 83}.
text_link: \texttt{19, 26, 27, 33, 36, 48, 54, 93, 97}.
text_pointer: \texttt{19, 20, 26, 31, 34, 50, 82, 90, 97}.
text_ptr: \texttt{19, 20, 21, 54, 83, 103}.
This can’t happen: \texttt{12}.
tok_mem: \texttt{2, 19, 20, 21, 26, 29, 31, 32, 83, 103}.
tok_mem_end: \texttt{20, 29, 82}.
tok_ptr: \texttt{19, 20, 21, 29, 82, 83, 103}.
tok_start: \texttt{19, 21, 26, 31, 33, 34, 36, 50, 54, 83}.
toupper: \texttt{89}.
transit: \texttt{57, 58, 59, 102}.
transit_code: \texttt{62, 63, 75, 100}.
transit_length: \texttt{57, 102}.
true: \texttt{4, 38, 42, 54, 64, 67, 69, 73, 74, 77, 79, 83, 86, 90, 100, 114, 115}.
try_loc: \texttt{87}.
uint16_t: \texttt{3, 4}.
uint8_t: \texttt{3, 4}.
unbreakable: \texttt{42, 55}.
Unrecognized escape sequence: \texttt{89}.
update_terminal: \texttt{15, 43, 48, 50, 90}.
Use @1 in limbo...: \texttt{75}.
use_language: \texttt{14}.
verbatim: \texttt{38, 42, 54, 55}.
Verbatim string didn’t end: \texttt{81}.
versionstring: \texttt{1, 116}.
web_file: \texttt{7}.
web_file_name: \texttt{7}.
web_file_open: \texttt{7, 48}.
wrap_up: \texttt{2, 13}.
write loop: \texttt{48}.
Writing the output...: \texttt{48}.
x: \texttt{29, 108}.
x_size: \texttt{108}.
xisalpha: \texttt{6, 102}.
xisdigit: \texttt{6, 69, 73, 89, 102}.
xislower: \texttt{6}.
ixispace: \texttt{6, 69, 79, 102}.
xisupper: \texttt{6}.
xisxdigit: \texttt{6, 73, 89, 102}.
y: \texttt{108}.
y_size: \texttt{108}.
〈Case of a section number 60〉 Used in section 55.
〈Case of an identifier 59〉 Used in section 55.
〈Cases like 1 = 56〉 Used in section 55.
〈Common code for C\textsc{weave} and C\textsc{tangle} 3*, 5*, 6*, 7*, 9*, 10*, 12*, 14*, 15*, 116*〉 Used in section 1*.
〈Compare the temporary output to the previous output 108*〉 Used in section 107*.
〈Compress two-symbol operator 71〉 Used in section 69.
〈Copy a string or verbatim construction or numerical constant 88*〉 Used in section 86*.
〈Copy an ASCII constant 89*〉 Used in section 86*.
〈Create the primary output depending on the comparison 109*〉 Used in section 106*.
〈Create the secondary output depending on the comparison 111*〉 Used in sections 110*, 112*, 113*, and 114*.
〈Expand section $a - 24000$, goto restart 40*〉 Used in section 38.
〈Get a constant 73〉 Used in section 69.
〈Get a string 74*〉 Used in section 69.
〈Get an identifier 72〉 Used in section 69.
〈Get control code and possible section name 75*〉 Cited in section 92. Used in section 69.
〈If end of name or erroneous nesting, break 80*〉 Used in section 79*.
〈If it’s not there, add cur_section_name to the output file stack, or complain we’re out of room 47*〉 Used in section 77.
〈If section is not being defined, continue 94〉 Used in section 90.
〈In cases that a is a non-char token (identifier, section_name, etc.), either process it and change a to a byte that should be stored, or continue if a should be ignored, or goto done if a signals the end of this replacement text 86*〉 Used in section 83*.
〈Include files 4*〉 Used in section 1*.
〈Initialize the output stacks 33〉 Used in section 48*.
〈Insert the line number into tok_mem 85〉 Used in sections 69, 83*, and 86*.
〈Insert the section number into tok_mem 96〉 Used in section 95.
〈Open the intermediate output file 105*〉 Used in section 50*.
〈Output macro definitions if appropriate 51〉 Used in section 48*.
〈Private variables 20*, 26, 32, 37, 42, 45, 52, 57, 62, 66, 68, 82*〉 Used in section 1*.
〈Put section name into section_text 79*〉 Used in section 77.
〈Read in transliteration of a character 102*〉 Used in section 100*.
〈Redirect temporary output to /dev/null 114*〉 Used in section 110*.
〈Redirect temporary output to /dev/\texttt{stderr} 113*〉 Used in section 110*.
〈Redirect temporary output to /dev/\texttt{stdout} 112*〉 Used in section 110*.
〈Scan a definition 93*〉 Used in section 90.
〈Scan a verbatim string 81*〉 Used in section 75*.
〈Scan an ASCII constant 76*〉 Used in section 75*.
〈Scan the C part of the current section 95〉 Used in section 90.
〈Scan the section name and make cur_section_name point to it 77〉 Used in section 75*.
〈Set initial values 21, 23, 27, 46, 58, 63, 78, 117*〉 Used in section 2*.
〈Set up the comparison of temporary output 107*〉 Used in sections 106* and 110*.
〈Setup system redirection 115*〉 Used in sections 112* and 113*.
〈Skip ahead until next\_control corresponds to @d, @<, @, @, or the like 92〉 Used in section 90.
〈Typedef declarations 19, 31*〉 Used in section 1*.
〈Update the data structure so that the replacement text is accessible 97〉 Used in section 95.
〈Update the primary result when it has changed 106*〉 Used in section 50*.
〈Update the secondary results when they have changed 110*〉 Used in section 50*.
〈Was an ‘@’ missed here? 87*〉 Used in section 86*.
〈Write all the named output files 50*〉 Used in section 48*.