Common code for CTANGLE and CWEAVE

(Version 4.2 [TeX Live])
1. **Introduction.** This file contains code common to CTANGLE, CWEAVE, and CTWILL, which roughly concerns the following problems: character uniformity, input routines, error handling and parsing of command line. We have tried to concentrate in this file all the system dependencies, so as to maximize portability.

In the texts below we will sometimes use CWEB to refer to any of the three component programs, if no confusion can arise.

The file begins with a few basic definitions.

(Include files 3*)
(Preprocessor definitions)
(Common code for CWEAVE and CTANGLE 2*)
(Global variables 18*)
(Predelaration of procedures 7*)

2. The details will be filled in due course. The interface of this module is included first. It is also used by the main programs.

First comes general stuff:

(Common code for CWEAVE and CTANGLE 2*) \equiv
typedef bool boolean;
typedef uint8_t eight_bits;
typedef uint16_t sixteen_bits;
typedef enum {
  ctangle, cweave, ctwill
} cweb;
extern cweb program; ▷ CTANGLE or CWEAVE or CTWILL? ◁
extern int phase; ▷ which phase are we in? ◁

See also sections 4*, 5*, 6*, 8*, 9*, 11*, 13*, and 14*.

This code is used in section 1*.

3. You may have noticed that almost all "strings" in the CWEB sources are placed in the context of the \_\_ \texttt{gettext} macro. This is just a shortcut for the \texttt{gettext} function from the “GNU gettext utilities.” For systems that do not have this library installed, we wrap things for neutral behavior without internationalization.

\#define \_\_ \texttt{gettext}(S)
(Include files 3*) \equiv
\#ifndef HAVE_GETTEXT
\#define HAVE_GETTEXT 0
\#endif
\#if HAVE_GETTEXT
\#include <libintl.h>
\#else
\#define gettext(A) A
\#endif
\#include <ctype.h> ▷ definition of isalpha, isdigit and so on ◁
\#include <stdbool.h> ▷ definition of bool, true and false ◁
\#include <stddef.h> ▷ definition of ptrdiff_t ◁
\#include <stdint.h> ▷ definition of uint8_t and uint16_t ◁
\#include <stdlib.h> ▷ definition of getenv and exit ◁
\#include <stdio.h> ▷ definition of printf and friends ◁
\#include <string.h> ▷ definition of strlen, strcmp and so on ◁

See also sections 89*, 91*, and 94*.

This code is used in section 1*. 
4* Code related to the character set:

```c
#define and_and 04  ▷ '&&'; corresponds to MIT's \&
#define lt_lt 020  ▷ '<<'; corresponds to MIT's ⟨
#define gt_gt 021  ▷ '>>'; corresponds to MIT's ⊳
#define plus_plus 013  ▷ '++'; corresponds to MIT's ↑
#define minus_minus 01  ▷ '--'; corresponds to MIT's ↓
#define minus_gt 031  ▷ '->'; corresponds to MIT's →
#define non_eq 032  ▷ '!='; corresponds to MIT's ≠
#define lt_eq 034  ▷ '<='; corresponds to MIT's ≤
#define gt_eq 035  ▷ '>='; corresponds to MIT's ≥
#define eq 036  ▷ '=='; corresponds to MIT's ≡
#define or_or 037  ▷ '||'; corresponds to MIT's ∨
#define dot_dot_dot 016  ▷ '...'; corresponds to MIT's ∞
#define colon_colon 06  ▷ ':'; corresponds to MIT's ∈
#define period_ast 026  ▷ '.*'; corresponds to MIT's ⊗
#define minus_ast 027  ▷ '-*'; corresponds to MIT's ↔
```

(Common code for CWEAVE and CTANGLE 2*)

5* Code related to input routines:

```c
#define xisalpha(c) (isalpha((eight_bits) c) ∧ ((eight_bits) c < 0200))
#define xisdigit(c) (isdigit((eight_bits) c) ∧ ((eight_bits) c < 0200))
#define xisspace(c) (isspace((eight_bits) c) ∧ ((eight_bits) c < 0200))
#define xislower(c) (islower((eight_bits) c) ∧ ((eight_bits) c < 0200))
#define xisupper(c) (isupper((eight_bits) c) ∧ ((eight_bits) c < 0200))
#define xisxdigit(c) (isxdigit((eight_bits) c) ∧ ((eight_bits) c < 0200))
```

(Common code for CWEAVE and CTANGLE 2*)
6* Code related to file handling:

```c
#define max_file_name_length 1024
#define cur_file_file[include_depth]  current file
#define cur_file_file_name[include_depth]  current file name
#define cur_line line[include_depth]  number of current line in current file
#define web_file_file[0]  main source file
#define web_file_file_name[0]  main source file name

extern int include_depth;  current level of nesting
extern FILE *file[];  stack of non-change files
extern FILE *change_file;  change file
extern char file_name[include_depth][max_file_name_length];  stack of non-change file names
extern char change_file_name[];  name of change file
extern char check_file_name[];  name of check file
extern int line[];  number of current line in the stacked files
extern int change_line;  number of current line in change file
extern int change_depth;  where \@y originated during a change
extern boolean input_has Ended;  if there is no more input
extern boolean change_pending;  if the current line is from change_file
extern boolean web_file_open;  if the web file is being read
```

7* (Predeclaration of procedures 7*)

```c
extern boolean get_line(void);  inputs the next line
extern void check_complete(void);  checks that all changes were picked up
extern void reset_input(void);  initialize to read the web file and change file
```

See also sections 10*, 12*, 15*, 24, 28, 33, 55, 64, 76, and 96*.
This code is used in section 1*.

8* Code related to section numbers:

```c
extern sixteen_bits section_count;  the current section number
extern boolean changed_section[];  is the section changed?
extern boolean change_pending;  is a decision about change still unclear?
extern boolean print_where;  tells \textsc{ctangle} to print line and file info
```
Common code for CTANGLE and CWEAVE (4.2 \LaTeX)

§ 9

Code related to identifier and section name storage:

\#define length(c) \((\text{size}_t)((c + 1)\cdot \text{byte}_\text{start} - (c)\cdot \text{byte}_\text{start})\) \triangleright the length of a name \triangleleft

\#define print_id(c) \text{term_write}((\text{byte}_\text{start}, \text{length}(c))) \triangleright print identifier \triangleleft

\#define \text{link} \text{link} \triangleright left link in binary search tree for section names \triangleleft

\#define rlink \text{dummy}\cdot\text{Rlink} \triangleright right link in binary search tree for section names \triangleleft

\#define root \text{name-dir-\text{rlink}} \triangleright the root of the binary search tree for section names \triangleleft

\langle Common code for CWEAVE and CTANGLE \rangle +

\begin{verbatim}
typedef struct name_info {
  char *byte_start; \triangleright beginning of the name in byte_mem \triangleleft
  struct name_info *link;
  union {
    struct name_info *Rlink; \triangleright right link in binary search tree for section names \triangleleft
    char Ilk; \triangleright used by identifiers in CWEAVE only \triangleleft
  } dummy;
  void *equiv_or_xref; \triangleright info corresponding to names \triangleleft
} name_info; \triangleright contains information about an identifier or section name \triangleleft

typedef name_info *name_pointer; \triangleright pointer into array of name_infos \triangleleft

typedef name_pointer *hash_pointer;

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

\langle Predeclaration of procedures \rangle +

\begin{verbatim}
extern boolean names_match(name_pointer, const char *, size_t, eight_bits);
extern name_pointer id_lookup(const char *, const char *, char);
  \triangleright looks up a string in the identifier table \triangleleft
extern name_pointer section_lookup(char *, char *, int); \triangleright finds section name \triangleleft
extern void init_node(name_pointer);
extern void init_p(name_pointer, eight_bits);
extern void print_prefix_name(name_pointer);
extern void print_section_name(name_pointer);
extern void sprint_section_name(char *, name_pointer);
\end{verbatim}
§11 Common code for CTANGLE and CWEAVE (4.2 [T\LaTeX Live])

INTRODUCTION

11* 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;
}
#endif
```

```c
#define confusion(s) fatal("\!\!\!\!\!\!\!\!\!\!", s)
```

(From Common code for CWEAVE and CTANGLE 2*) +≡

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

12* (Predeclaration of procedures ?*) +≡

```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 */
```

13* 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_stats_flags[\'s\']  /* should statistics be printed at end of run? */
#define show_happiness_flags[\'h\']  /* should lack of errors be announced? */
#define temporary_output_flags[\'t\']  /* should temporary output take precedence? */
#define make_xrefs_flags[\'x\']  /* should cross references be output? */
```

(From Common code for CWEAVE and CTANGLE 2*) +≡

```c
extern int argc;  /* 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 */
```

14* Code related to output:

```c
#define update_terminal fflush(stdout)  /* empty the terminal output buffer */
#define new_line putchar(\'\n\')
#define putchar putchar
#define term_write(a, b) fflush(stdout), fwrite(a, sizeof(char), b, stdout)
#define C_printf(c, a) fprintf(C_file, c, a)
#define C_putc(c) putc(c, C_file)  /* isn't C wonderfully consistent? */
```

(From Common code for CWEAVE and CTANGLE 2*) +≡

```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 */
```
15* The procedure that gets everything rolling:

(Predeclaration of procedures 7*) +≡

extern void common_init( void);
extern void print_stats( void);
extern void cb_show_banner( void);

16* The following parameters were sufficient in the original WEB to handle \TeX, so they should be sufficient for most applications of CWEB.

#define max_bytes 1000000 ▶ the number of bytes in identifiers, index entries, and section names ◁
#define max_toks 1000000 ▶ number of bytes in compressed C code ◁
#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 ◁
#define max_texts 10239 ▶ number of replacement texts, must be less than 10240 ◁
#define longest_name 10000 ▶ file and section names and section texts shouldn’t be longer than this ◁
#define stack_size 500 ▶ number of simultaneous levels of macro expansion ◁
#define buf_size 1000 ▶ maximum length of input line, plus one ◁
#define long_buf_size (buf_size + longest_name) ▶ for CWEAVE ◁

17* End of COMMON interface.

18* In certain cases CTANGLE and CWEAVE should do almost, but not quite, the same thing. In these cases we’ve written common code for both, differentiating between the two by means of the global variable program. And CTWILL adds some extra twists.

(Global variables 18*) ≡

cweb program; ▶ CTANGLE or CWEAVE or CTWILL? ◁

See also sections 19, 21, 22, 25*, 26, 37, 43, 44, 46, 65, 73*, 83*, 86*, and 87*.

This code is used in section 1*.

20* There’s an initialization procedure that gets both CTANGLE and CWEAVE off to a good start. We will fill in the details of this procedure later.

void common_init( void)
{
  ⟨Initialize pointers 45⟩
  ⟨Set up PROGNAME feature and initialize the search path mechanism 92*⟩
  ⟨Set locale and bind language catalogs 90*⟩
  ⟨Set the default options common to CTANGLE and CWEAVE 74*⟩
  ⟨Scan arguments and open output files 84*⟩
}
23* In the unlikely event that your standard I/O library does not support `feof`, `getc`, and `ungetc` you may have to change things here.

```java
static boolean input_ln(FILE *fp) {  // copies a line into buffer or returns false
    register int c ← EOF;  // character read; initialized so some compilers won’t complain
    register char *k;   // where next character goes
    if (feof(fp)) return false;  // we have hit end-of-file
    limit ← k ← buffer;  // beginning of buffer
    while (k ≤ buffer_end ∧ (c ← getc(fp)) ≠ EOF ∧ c ≠ 'n')
        if ((c ← getc(fp)) ≠ EOF ∧ c ≠ 'r') limit ← k;
        if (k > buffer_end)
            if ((c ← getc(fp)) ≠ EOF ∧ c ≠ 'n') {
                ungetc(c, fp); loc ← buffer; err_print("!
Input_line too long");
            }
    if (c ≡ EOF ∧ limit ≡ buffer) return false;  // there was nothing after the last newline
    return true;
}
```

25* Now comes the problem of deciding which file to read from next. Recall that the actual text that CWEB should process comes from two streams: a web file, which can contain possibly nested include commands @i, and a change file, which might also contain includes. The web file together with the currently open include files form a stack file, whose names are stored in a parallel stack file name. The boolean changing tells whether or not we’re reading from the change file.

The line number of each open file is also kept for error reporting and for the benefit of CTANGLE.

(Global variables 18*) +≡

int include_depth;  // current level of nesting
FILE *file[include_depth];  // stack of non-change files
FILE *change_file;  // change file
char file_name[include_depth][max_file_name_length];  // stack of non-change file names
char change_file_name[include_depth][max_file_name_length];  // name of change file
int line[include_depth];  // number of current line in the stacked files
int change_line;  // number of current line in change file
int change_depth;  // where @y originated during a change
boolean input_has Ended;  // if there is no more input
boolean changing;  // if the current line is from change_file
boolean web_file_open ← false;  // if the web file is being read
```
29* While looking for a line that begins with \@x in the change file, we allow lines that begin with \@, as long as they don’t begin with \@y, \@z, or \@i (which would probably mean that the change file is fouled up).

(Skip over comment lines in the change file; return if end of file 29*)

\[
\text{while (true) } \{
\text{change_line}++; \\
\text{if } (\neg \text{input_in(change_file)}) \text{ return;}
\}
\]

This code is used in section 27.

30* Here we are looking at lines following the \@x.

(Skip to the next nonblank line; return if end of file 30*)

\[
\text{do } \{
\text{change_line}++; \\
\text{if } (\neg \text{input_in(change_file)}) \{ \\
\text{err_print}(."!Missing\@x in change file"); \text{ return;}
\}
\}
\]

This code is used in section 27.
The following procedure is used to see if the next change entry should go into effect; it is called only when changing is false. The idea is to test whether or not the current contents of buffer matches the current contents of change_buffer. If not, there’s nothing more to do; but if so, a change is called for: All of the text down to the @y is supposed to match. An error message is issued if any discrepancy is found. Then the procedure prepares to read the next line from change_file.

When a match is found, the current section is marked as changed unless the first line after the @x and after the @y both start with either ‘@*’ or ‘@␣’ (possibly preceded by whitespace).

This procedure is called only when buffer < limit, i.e., when the current line is nonempty.

```c
#define if_section_start_make_pending(b) {
  *limit ← ’!’;
  for (loc ← buffer; xisspace(*loc); loc ++);
  *limit ← ’ ’;
  if (*loc ≡ ’@’ ∧ (xisspace(*(loc + 1)) ∨ *(loc + 1) ≡ ’*’)) change_pending ← b;
}

static void check_change(void)▷switches to change_file if the buffers match◁{
  int n ← 0;▷the number of discrepancies found◁
  if (lines_dont_match) return;
  change_pending ← false;
  if (!changed_section[section_count]) {
    if_section_start_make_pending(true);
    if (!change_pending) changed_section[section_count] ← true;
  }
  while (true) {
    changing ← true; print_where ← true; change_line ++;
    if (!input_line(change_file)) { 
      err.print("!Change_file ended before @y"); change_limit ← change_buffer;
      changing ← false; return;
    }
    if (limit > buffer + 1 ∧ buffer[0] ≡ ’@’) {
      char xyz_code ← xisupper(buffer[1]) ? tolower((eight_bits buffer[1]) : buffer[1]);
      (If the current line starts with @y, report any discrepancies and return 34*)
    } (Move buffer and limit to change_buffer and change_limit 31)
    changing ← false; cur_line ++;
    while (!input_line(cur_file)) {▷pop the stack or quit◁
      if (include_depth ≡ 0) {
        err.print("!CWEB_file ended during a change"); input_hasEnded ← true; return;
      }
      include_depth --; cur_line ++;
    }
    if (lines_dont_match) n ++;
  }
}
If the current line starts with `@y`, report any discrepancies and `return 34`.

```c
if (xyz_code == 'x' ∨ xyz_code == 'z') {
  loc ← buffer + 2;  err_print(\"Where is the matching @y?\")
}  else if (xyz_code == 'y') {
  if (n > 0) {
    loc ← buffer + 2;  printf("\n! Hmm... %d of the preceding lines failed to match\")
  }
  change_depth ← include_depth;  return;
}
```

This code is used in section 32.*.

36* The following code opens the input files.

```c
if ((found_filename ← kpsfind_cweb(web_file_name)) ≡ Λ ∨
    (web_file ← fopen(found_filename, "r")) ≡ Λ) {
  fatal(\"Cannot open input file\", web_file_name);
}  else if (strlen(found_filename) < max_file_name_length) {  ▷ Copy name for #line directives. ◁
  if (strcmp(web_file_name, found_filename))
    strcpy(web_file_name, found_filename + ((strncmp(found_filename, ".", 2) ≡ 0) ? 2 : 0));
    free(found_filename);
  else fatal(\"Filename too long\", found_filename);
web_file_open ← true;
if ((found_filename ← kpsfind_cweb(change_file_name)) ≡ Λ ∨
    (change_file ← fopen(found_filename, "r")) ≡ Λ) {
  fatal(\"Cannot open change file\", change_file_name);
}  else if (strlen(found_filename) < max_file_name_length) {  ▷ Copy name for #line directives. ◁
  if (strcmp(change_file_name, found_filename))
    strcpy(change_file_name, found_filename + ((strncmp(found_filename, ".", 2) ≡ 0) ? 2 : 0));
    free(found_filename);
  else fatal(\"Filename too long\", found_filename);
```

This code is used in section 35.
38* boolean get_line(void) ▷ inputs the next line ◄
{
    restart:
    if (changing ∧ include_depth ≡ change_depth)
        ⟨Read from change_file and maybe turn off changing 41*⟩
    if (¬changing ∨ include_depth > change_depth) {
        ⟨Read from cur_file and maybe turn on changing 40⟩
        if (changing ∧ include_depth ≡ change_depth) goto restart;
    }
    if (input_has Ended) return false;
    loc ← buffer; *limit ← ' ';  
    if (buffer[0] ≡ '@' ∧ (buffer[1] ≡ 'i' ∨ buffer[1] ≡ 'I')) {
        loc ← buffer + 2; *limit ← ' ';
        while (*loc ≡ ' ' ∨ *loc ≡ '	') loc ++;
        if (loc ≥ limit) {
            err_print(("!_Include_file_name_not_given")); goto restart;
        }
        if (include_depth ≥ max_include_depth - 1) {
            err_print(("!_Too_many_nested_includes")); goto restart;
        }
        include_depth ++;  ◄ push input stack ◄
        ⟨Try to open include file, abort push if unsuccessful, go to restart 39*⟩
    }
    return true;
}
39* When an \@i line is found in the \texttt{cur_file}, we must temporarily stop reading it and start reading from the named include file. The \@i line should give a complete file name with or without double quotes. The actual file lookup is done with the help of the KPATHSEA library; see section \langle File lookup with KPATHSEA 91 \rangle for details. The remainder of the \@i line after the file name is ignored.

\begin{verbatim}
#define too_long()
{
    include_depth--; err_print(\texttt{!Include file name too long}); goto restart;
}

(Try to open include file, abort push if unsuccessful, go to restart 39*) \equiv
{
    char *cur_file_name_end ← cur_file_name + max_file_name_length - 1;
    char *k ← cur_file_name;
    if (*loc \equiv \\
        \{
            loc++; \\
            while (*loc \equiv \\
                \{  \\
                    k ← cur_file_name_end + 1; \quad \triangleright \text{unmatched quote is 'too long'} <
                \}
            else
                \{
                    while (*loc \neq \\
                        \{'\n                    \}
                \}
            if (k > cur_file_name_end) too_long();
            \}
        \}
    \}
    if ((found_filename ← kpse_find_cweb(cur_file_name)) \neq \Lambda \land
        \{  \\
            cur_file ← fopen(found_filename, "r") \neq \Lambda \}
    \}
        \{ \quad \triangleright \text{Copy name for \#line directives.} \} <
        \{  \\
            if (strlen(found_filename) < max_file_name_length) \\
            \}
            if (strcmp(cur_file_name, found_filename) \equiv 0) \
                \{  \\
                    strcpy(cur_file_name, found_filename + ((strncmp(found_filename, ".", 2) \equiv 0) ? 2 : 0));
                \}
        \}
    \}
        \} <
        \}
    \}
}

This code is used in section 38*.
\end{verbatim}
41* (Read from change_file and maybe turn off changing 41*) \equiv
\{ change_line++;
  if (!input_line(change_file)) {
    err_print(\("!Change_file\_ended\_without\_@z\"\));
    buffer[0] \leftarrow '0'; buffer[1] \leftarrow 'z';
    limit \leftarrow buffer + 2;
  }
  if (limit > buffer) { \triangleright check if the change has ended \triangleleft
    if (change_pending) {
      if_section_start_make_pending(false);
      if (change_pending) {
        changed_section[section_count] \leftarrow true; change_pending \leftarrow false;
      }
    }
    *limit \leftarrow '\_';
    if (buffer[0] \equiv '0') {
      if (xisupper(buffer[1])) buffer[1] \leftarrow tolower((\text{eight_bits}) buffer[1]);
      if (buffer[1] \equiv 'x' \lor buffer[1] \equiv 'y') {
        loc \leftarrow buffer + 2; err_print(\("!Where\_is\_the\_matching\_@z?\"\));
      } else if (buffer[1] \equiv 'z') {
        prime_the_change_buffer(); changing \leftarrow \neg changing; print_where \leftarrow true;
      }
    }
  }
}\}

This code is used in section 38*.

42* At the end of the program, we will tell the user if the change file had a line that didn’t match any relevant line in web_file.

\texttt{void check_complete(void)} \equiv
\{ if (change_limit \neq change_buffer) { \triangleright changing is false \triangleleft
  strncpy(buffer, change_buffer, (\text{size_t})(change_limit - change_buffer + 1));
  limit \leftarrow buffer + (\text{ptrdiff_t})(change_limit - change_buffer); changing \leftarrow true;
  change_depth \leftarrow include_depth; loc \leftarrow buffer;
  err_print(\("!Change_file\_entry\_did\_not\_match\"\));
}\}

51* The information associated with a new identifier must be initialized in a slightly different way in CWEAVE than in CTANGLE; hence the init_p procedure.

\texttt{(Enter a new name into the table at position p 51*) \equiv}
\{ if (byte_ptr + l > byte_mem_end) overflow(\(_"byte\_memory\"\));
  if (name_ptr \geq name_dir_end) overflow(\(_"name\"\));
  strncpy(byte_ptr, first, l); (++name_ptr)-byte_start \leftarrow byte_ptr += l; init_p(p, t);
}\}

This code is used in section 48.
Adding a section name to the tree is straightforward if we know its parent and whether it’s the rlink or llink of the parent. As a special case, when the name is the first section being added, we set the “parent” to Λ. When a section name is created, it has only one chunk, which however may be just a prefix; the full name will hopefully be unveiled later. Obviously, prefix_length starts out as the length of the first chunk, though it may decrease later.

The information associated with a new node must be initialized differently in CWEAVE and CTANGLE; hence the init_node procedure, which is defined differently in cweave.w and ctangle.w.

```c
static name_pointer add_section_name(  // install a new node in the tree
    name_pointer par,  // parent of new node
    int c,  // right or left?
    char *first,  // first character of section name
    char *last,  // last character of section name, plus one
    int ispref)  // are we adding a prefix or a full name?
{
    name_pointer p ← name_ptr;  // new node
    char *s ← first_chunk(p);
    int name_len ← (int)((last − first) + ispref);  // length of section name
    if (s + name_len > byte_mem_end) overflow(_."byte_memory");
    if (name_ptr + 1 ≥ name_dir_end) overflow(_."name");
    (++name_ptr)−byte_start ← byte_ptr ← s + name_len;
    if (ispref) {
        *(byte_ptr − 1) ← ' '; name_len ←−; name_ptr−link ← name_dir;
        (++name_ptr)−byte_start ← byte_ptr;
    }
    set_prefix_length(p, name_len); strncpy(s, first, name_len);
    p−llink ← Λ; p−rlink ← Λ; init_node(p);
    return par ≡ Λ ? (root ← p) : c ≡ less ? (par−llink ← p) : (par−rlink ← p);
}
```

```c
static void extend_section_name(name_pointer p,  // name to be extended
    char *first,  // beginning of extension text
    char *last,  // one beyond end of extension text
    int ispref)  // are we adding a prefix or a full name?
{
    char *s;
    name_pointer q ← p + 1;
    int name_len ← (int)((last − first) + ispref);
    if (name_ptr ≥ name_dir_end) overflow(_."name");
    while (q−link ≠ name_dir) q ← q−link;
    q−link ← name_ptr; s ← name_ptr−byte_start; name_ptr−link ← name_dir;
    if (s + name_len > byte_mem_end) overflow(_."byte_memory");
    (++name_ptr)−byte_start ← byte_ptr ← s + name_len; strncpy(s, first, name_len);
    if (ispref) *(byte_ptr − 1) ← ' ';}
```
A legal new name matches an existing section name if and only if it matches the shortest prefix of that section name. Therefore we can limit our search for matches to shortest prefixes, which eliminates the need for chunk-chasing at this stage.

This code is used in section 59.

Although error messages are given in anomalous cases, we do return the unique best match when a discrepancy is found, because users often change a title in one place while forgetting to change it elsewhere.

This code is used in section 59.
REPORTING ERRORS TO THE USER

Common code for CTANGLE and CWEAVE (4.2 \[\TeX\ Live\])

The error locations can be indicated by using the global variables $loc$, $cur\_line$, $cur\_file\_name$ and $changing$, which tell respectively the first unlooked-at position in $buffer$, the current line number, the current file, and whether the current line is from $change\_file$ or $cur\_file$. This routine should be modified on systems whose standard text editor has special line-numbering conventions.

{\texttt{printf error location based on input buffer} \footnote{67*}} \equiv

\begin{verbatim}
if (changing \land include\_depth \equiv change\_depth)
  printf (\{ \text{\texttt{. \\textbackslash{n}} \texttt{(l.\textbackslash{\%d}of\texttt{\textbackslash{\%s}\textbackslash{n})}}} \texttt{\textbackslash{n}} \}), cur\_line);
else if (include\_depth \equiv 0)
  printf (\{ \text{\texttt{. \\textbackslash{n}}} \texttt{(l.\textbackslash{\%d})}} \texttt{\textbackslash{n}} \}}, cur\_line);
else
  printf (\{ \text{\texttt{. \\textbackslash{n}}} \texttt{(l.\textbackslash{\%d}of\texttt{\textbackslash{\%s}}} \texttt{\textbackslash{n})}}} \texttt{\textbackslash{n}} \}}, cur\_line, cur\_file\_name);
\end{verbatim}

\begin{verbatim}
l \leftarrow (loc \geq limit ? limit : loc);
if (l > buffer) {
  for (k \leftarrow buffer; k < l; k++)
    if (*k \equiv 't') 
      putchar ('\t');
    else
      putchar (*k); \quad \text{\texttt{\textbackslash{n}}};
  for (k \leftarrow buffer; k < l; k++)
    putchar ('\t'); \quad \text{\texttt{\textbackslash{n}}};
}
for (k \leftarrow l; k < limit; k++)
  putchar (*k); \quad \text{\texttt{\textbackslash{n}}};
if (*limit \equiv 't')
  putchar ('\t'); \quad \text{\texttt{\textbackslash{n}}};
  \quad \text{\texttt{\textbackslash{n}}};
\end{verbatim}

\begin{verbatim}
\} \quad \text{\texttt{\textbackslash{n}}};
\} \quad \text{\texttt{\textbackslash{n}}};
\end{verbatim}

This code is used in section 66.

\footnote{68*}{When no recovery from some error has been provided, we have to wrap up and quit as graciously as possible. This is done by calling the function \texttt{wrap\_up} at the end of the code.

CTANGLE and CWEAVE have their own notions about how to print the job statistics. See the function(s) \texttt{print\_stats} in the interface above and in the index.

On multi-tasking systems like the AMIGA it is very convenient to know a little bit more about the reasons why a program failed. The four levels of return indicated by the \texttt{history} value are very suitable for this purpose. Here, for instance, we pass the operating system a status of 0 if and only if the run was a complete success. Any warning or error message will result in a higher return value, so that AREXX scripts can be made sensitive to these conditions.

\begin{verbatim}
#define RETURN_OK 0 \quad \text{\texttt{\textbackslash{n}}};
#define RETURN_WARN 5 \quad \text{\texttt{\textbackslash{n}}};
#define RETURN_ERROR 10 \quad \text{\texttt{\textbackslash{n}}};
#define RETURN_FAIL 20 \quad \text{\texttt{\textbackslash{n}}};
\end{verbatim}

\begin{verbatim}
int wrap\_up(void)
{
  if (show\_progress) new\_line;
  if (show\_stats) print\_stats(); \quad \text{\texttt{\textbackslash{n}}};
  \langle \text{Print the job history} \footnote{69*} \rangle
  \langle\text{Remove the temporary file if not already done} \footnote{88*} \rangle
  switch (history) {
    case harmless\_message: return RETURN\_WARN;
    case error\_message: return RETURN\_ERROR;
    case fatal\_message: return RETURN\_FAIL;
    default: return RETURN\_OK;
  }
}
\end{verbatim}
§69  Common code for CTANGLE and CWEAVE (4.2 [\TeX{} Live])  REPORTING ERRORS TO THE USER  17

69*  (Print the job history 69*) ≡
switch (history) {
  case spotless:
    if (show_happiness) puts(_("(No errors were found.)"));
    break;
  case harmless_message: puts(_("(Did you see the warning message above?)"); break;
  case error_message: puts(_("(Pardon me, but I think I spotted something wrong.)"); break;
  case fatal_message: puts(_("(That was a fatal error, my friend.)");
}   ▷  there are no other cases ◁
This code is used in section 68*.

71*  An overflow stop occurs if CWEB’s tables aren’t large enough.

void overflow(const char *t)
{
  printf(_("\n! Sorry, %s capacity exceeded"), t); fatal("", "");
}
73* Command line arguments. The user calls \texttt{CWEAVE} and \texttt{CTANGLE} with arguments on the command line. These are either file names or flags to be turned off (beginning with "-" ) or flags to be turned on (beginning with "+"). The following globals are for communicating the user’s desires to the rest of the program. The various file name variables contain strings with the names of those files. Most of the 128 flags are undefined but available for future extensions.

\begin{verbatim}
\langle Global variables \rangle +≡
  int argc;         \triangleright copy of \texttt{ac} parameter to \texttt{main} \triangleright
  char **argv;      \triangleright copy of \texttt{av} parameter to \texttt{main} \triangleright
  char C_file_name[max_file_name_length]; \triangleright name of C_file \triangleright
  char tex_file_name[max_file_name_length]; \triangleright name of tex_file \triangleright
  char idx_file_name[max_file_name_length]; \triangleright name of idx_file \triangleright
  char scn_file_name[max_file_name_length]; \triangleright name of scn_file \triangleright
  char check_file_name[max_file_name_length]; \triangleright name of check_file \triangleright
  boolean flags[128]; \triangleright an option for each 7-bit code \triangleright
\end{verbatim}

74* The flags will be initially false. Some of them are set to true before scanning the arguments; if additional flags are true by default they should be set before calling \texttt{common_init}.

\begin{verbatim}
\langle Set the default options common to \texttt{CTANGLE} and \texttt{CWEAVE} \rangle +≡
  make_xrefs ← true;
  temporary_output ← true; \triangleright Check temporary output for changes \triangleright
  show_stats ← false;
\end{verbatim}

This code is used in section 20*.
We now must look at the command line arguments and set the file names accordingly. At least one file name must be present: the CWEB file. It may have an extension, or it may omit the extension to get ".w" added. The \TeX\ output file name is formed by replacing the CWEB file name extension by ".tex", and the C file name by replacing the extension by ".c", after removing the directory name (if any).

If there is a second file name present among the arguments, it is the change file, again either with an extension or without one to get ".ch". An omitted change file argument means that "/dev/null" or—on non-UNIX systems the contents of the compile-time variable \texttt{DEV\_NULL} (\TeX\ Live) or \texttt{\_DEV\_NULL} (Amiga)—should be used, when no changes are desired.

If there's a third file name, it will be the output file.

```c
static void scan_args(void)
{
    char *dot_pos;  ▷ position of '. ' in the argument ◀
    char *name_pos; ▷ file name beginning, sans directory ◀
    register char *s;  ▷ register for scanning strings ◀
    boolean found_web ← false, found_change ← false, found_out ← false;
    ▷ have these names been seen? ◀
    #if defined DEV\_NULL
        strncpy(change_file_name, DEV\_NULL, max_file_name_length − 2);
        change_file_name[max_file_name_length − 2] ← \'\0\';
    #elif defined _DEV\_NULL
        strncpy(change_file_name, _DEV\_NULL, max_file_name_length − 2);
        change_file_name[max_file_name_length − 2] ← \'\0\';
    #else
        strcpy(change_file_name, "/dev/null");
    #endif
    while (−−argc > 0) {
        if (((**(++argv) ≡ \'\−\' \∨ **argv ≡ \'\+\') \∧ *(argv + 1))) (Handle flag argument 80*)
            else {
                s ← name_pos ← argv; dot_pos ← Λ;
                while (**s) {
                    if (**s ≡ \'\−\') dot_pos ← s++;
                    else if (**s ≡ DIR\_SEPARATOR \∨ **s ≡ DEVICE\_SEPARATOR \∨ **s ≡ \'\/\')
                        dot_pos ← Λ, name_pos ← ++s;
                    else s++;
                }
                if (!found_web) (Make web_file_name, tex_file_name, and C_file_name 77*)
                else if (!found_change) (Make change_file_name from fname 78)
                else if (!found_out) (Override tex_file_name and C_file_name 79)
                else (Print usage error message and quit 81*)
            }
        else (Print usage error message and quit 81*)
    }

} 
```
We use all of *argv for the web_file_name if there is a ' ' in it, otherwise we add ".w". The other file names come from adding other things after the dot. We must check that there is enough room in web_file_name and the other arrays for the argument.

\begin{verbatim}
(Make web_file_name, tex_file_name, and C_file_name 77*)
{ 
    if (s – *argv > max_file_name_length – 5) \langle Complain about argument length 82*
    if (dot_pos == 0) sprintf(web_file_name, "%s.w", *argv);
    else {
        strcpy(web_file_name, *argv); *dot_pos = 0; \triangleright string now ends where the dot was \langle
    }
    printf(tex_file_name, "%s.tex", name_pos);
    \triangleright strip off directory name \langle
    sprintf(idx_file_name, "%s.idx", name_pos);
    sprintf(scn_file_name, "%s.scn", name_pos);
    sprintf(C_file_name, "%s.c", name_pos); found_web \leftarrow true;
}
\end{verbatim}

This code is used in section 75*.

\begin{verbatim}
#define flag_change (**argv \neq '−−')
\end{verbatim}

\begin{verbatim}
(Handle flag argument 80*)
{ 
    if (strcmp("−−help", *argv) \equiv 0 \lor strcmp("−−help", *argv) \equiv 0) \langle Display help message and exit 95*
    if (strcmp("−−version", *argv) \equiv 0 \lor strcmp("−−version", *argv) \equiv 0) 
        \langle Display version information and exit 98*
    if (strcmp("−−verbose", *argv) \equiv 0 \lor strcmp("−−verbose", *argv) \equiv 0) {
        show_banner \leftarrow show_progress \leftarrow show_happiness \leftarrow 1; continue;
    }
    if (strcmp("−−quiet", *argv) \equiv 0 \lor strcmp("−−quiet", *argv) \equiv 0) {
        show_banner \leftarrow show_progress \leftarrow show_happiness \leftarrow 0; continue;
    }
    for (dot_pos = *argv + 1; *dot_pos > '\0'; dot_pos++)
        if (*dot_pos \equiv 'v') {
            show_banner \leftarrow show_progress \leftarrow show_happiness \leftarrow true;
        }
    else if (*dot_pos \equiv 'q') {
        show_banner \leftarrow show_progress \leftarrow show_happiness \leftarrow false;
    }
    else if (*dot_pos \equiv 'd') {
        if (sscanf(++dot_pos, "%u", &kpathsea_debug) \neq 0) \langle Print usage error message and quit 81*
            while (isdigit(*dot_pos)) dot_pos++; \triangleright skip numeric part \langle
            dot_pos--; \triangleright reset to final digit \langle
        }
    else if (*dot_pos \equiv 'l') {
        use_language \leftarrow ++dot_pos; break;
    }
    else flags[(eight_bits) *dot_pos] \leftarrow flag_change;
}
\end{verbatim}

This code is used in section 75*.

\begin{verbatim}
(Print usage error message and quit 81*)
{ 
    cb_usage(program \equiv ctangle ? "ctangle" : program \equiv cweave ? "cweave" : "ctwill="%n
}
\end{verbatim}

This code is used in sections 75* and 80*.
\begin{verbatim}
82* (Complain about argument length 82*) ≡
  fatal("!\_Filename\_too\_long\n", *argv);
\end{verbatim}

This code is used in sections \texttt{77*}, \texttt{78}, and \texttt{79}. 

\section{Common code for CTANGLE and CWEAVE (4.2 \LaTeX Live)}
83* Output. Here is the code that opens the output file:

\[
\text{\texttt{\langle Global variables \ 18* \rangle \equiv}}
\]

\[
\text{\texttt{FILE *C_file; \quad \triangleright \text{ where output of CTANGLE goes \triangleright}}}
\]
\[
\text{\texttt{FILE *tex_file; \quad \triangleright \text{ where output of CWEAVE goes \triangleright}}}
\]
\[
\text{\texttt{FILE *idx_file; \quad \triangleright \text{ where index from CWEAVE goes \triangleright}}}
\]
\[
\text{\texttt{FILE *scn_file; \quad \triangleright \text{ temporary output file \triangleright}}}
\]
\[
\text{\texttt{FILE *check_file; \quad \triangleright \text{ where list of sections from CWEAVE goes \triangleright}}}
\]
\[
\text{\texttt{FILE *active_file; \quad \triangleright \text{ currently active file for CWEAVE output \triangleright}}}
\]
\[
\text{\texttt{char *found_filename; \quad \triangleright \text{ filename found by kpse\_find\_file \triangleright}}}
\]

84* \text{\texttt{(Scan arguments and open output files 84*}) \equiv}

\[
\text{\texttt{scan\_args();}}
\]
\[
\text{\texttt{if (program \equiv ctangle) \{} }
\]
\[
\text{\texttt{if ((C_file \leftarrow fopen(C\_file\_name, "a")) \equiv \Lambda) \ fatal(\text{"%!C\_Cannot\_open\_output\_file\_"}, C\_file\_name);}}
\]
\[
\text{\texttt{else fclose(C\_file); \quad \triangleright \text{ Test accessibility \triangleright}}}
\]
\[
\text{\texttt{strcpy(check\_file\_name, C\_file\_name);}}
\]
\[
\text{\texttt{if (check\_file\_name[0] \neq '\0') \{} }
\]
\[
\text{\texttt{char *dot\_pos \leftarrow strchr(check\_file\_name, '.');}}
\]
\[
\text{\texttt{if (dot\_pos \equiv \Lambda) strcat(check\_file\_name, "\.ttp");}}
\]
\[
\text{\texttt{else strcpy(dot\_pos, "\.ttp");}}
\]
\[
\text{\texttt{}}} \]
\[
\text{\texttt{if ((C_file \leftarrow fopen(check\_file\_name, "wb")) \equiv \Lambda)}}
\]
\[
\text{\texttt{fatal(\text{"%!C\_Cannot\_open\_output\_file\_"}, check\_file\_name);}}
\]
\[
\text{\texttt{}}} \]
\[
\text{\texttt{else \{} }
\]
\[
\text{\texttt{if ((tex_file \leftarrow fopen(tex\_file\_name, "a")) \equiv \Lambda)}}
\]
\[
\text{\texttt{fatal(\text{"%!C\_Cannot\_open\_output\_file\_"}, tex\_file\_name);}}
\]
\[
\text{\texttt{else fclose(tex_file); \quad \triangleright \text{ Test accessibility \triangleright}}}
\]
\[
\text{\texttt{strcpy(check\_file\_name, tex\_file\_name);}}
\]
\[
\text{\texttt{if (check\_file\_name[0] \neq '\0') \{} }
\]
\[
\text{\texttt{char *dot\_pos \leftarrow strchr(check\_file\_name, '.');}}
\]
\[
\text{\texttt{if (dot\_pos \equiv \Lambda) strcat(check\_file\_name, "\.wtp");}}
\]
\[
\text{\texttt{else strcpy(dot\_pos, "\.wtp");}}
\]
\[
\text{\texttt{}}} \]
\[
\text{\texttt{if ((tex_file \leftarrow fopen(check\_file\_name, "wb")) \equiv \Lambda)}}
\]
\[
\text{\texttt{fatal(\text{"%!C\_Cannot\_open\_output\_file\_"}, check\_file\_name);}}
\]
\[
\text{\texttt{}}} \]

This code is used in section 20*.
Extensions to \texttt{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 "\textbf{85. Index}."
Language setting. This global variable is set by the argument of the ‘+1’ (or ‘-1’) command-line option.

(Global variables 18*) \[=\]

```
const char *use_language \= "";
▷ prefix of cwebmac.tex in \TeX\ output ◁
```
87* User communication. The *scan_args* and *cb_show_banner* routines and the *bindtextdomain* argument string need a few extra variables.

```c
#define max_banner 50
#define PATH_SEPARATOR separators[0]
#define DIR_SEPARATOR separators[1]
#define DEVICE_SEPARATOR separators[2]

(char cb_banner[max_banner]);
(string texmf_locale);
#endif
#define SEPARATORS "://"
#endif
(char separators[]) ← SEPARATORS;
```
88* Temporary file output. Before we leave the program we have to make sure that the output files are correctly written.

( Remove the temporary file if not already done 88* )

\[
\begin{align*}
& \text{if ( } \text{C\_file} \text{ ) } fclose( \text{C\_file} ); \\
& \text{if ( } \text{tex\_file} \text{ ) } fclose( \text{tex\_file} ); \\
& \text{if ( } \text{check\_file} \text{ ) } fclose( \text{check\_file} ); \\
& \text{if ( } \text{strlen( } \text{check\_file\_name} \text{ )} \text{ ) } \triangleright \text{ Delete the temporary file in case of a break } \triangleleft \\
& \qquad \text{remove( } \text{check\_file\_name} \text{ );}
\end{align*}
\]

This code is used in section 68*.
Internationalization. If translation catalogs for your personal LANGUAGE are installed at the appropriate place, CTANGLE and CWEAVE will talk to you in your favorite language. Catalog \texttt{cweb} contains all strings from “plain CWEB,” catalog \texttt{cweb-tl} contains a few extra strings specific to the \TeX{} Live interface, and catalog \texttt{web2c-help} contains the “\texttt{--help}” texts for CTANGLE and CWEAVE.

If such translation files are not available, you may want to improve this system by checking out the sources and translating the strings in files \texttt{cweb.pot}, \texttt{cweb-tl.pot}, and \texttt{web2c-help.pot}, and submitting the resulting *.po files to the maintainers at \texttt{tex-k@tug.org}.

Note to maintainers: CWEB in \TeX{} Live generally does not set \texttt{HAVE_GETTEXT} at build-time, so i18n is “off” by default. If you want to create CWEB executables with NLS support, you have to recompile the \TeX{} Live sources with a positive value for \texttt{HAVE_GETTEXT} in \texttt{comm-w2c.h}. Also you have to “compile” the NLS catalogs provided for CWEB in the source tree with \texttt{msgfmt} and store the resulting .mo files at an appropriate place in the file system.

Plans for \TeX{} Live are to store NLS catalogs inside the “\TeX{} Directory Structure” (TDS) and look them up with the help of the configuration variable “\texttt{TEXMFLOCALEDIR},” which should contain a single absolute path definition. Below we use the KPATHSEA function \texttt{kpse_var_expand} to evaluate this variable from various origins and redirect the “GNU gettext utilities” to a possibly different location than the canonical /usr/share/locale.

There are several ways to set \texttt{TEXMFLOCALEDIR}:

(a) a user-set environment variable \texttt{TEXMFLOCALEDIR}
   (overridden by \texttt{TEXMFLOCALEDIR_cweb});
(b) a line in KPATHSEA configuration file \texttt{texmf.cnf},
   e.g., \texttt{TEXMFLOCALEDIR=$TEXMFMAIN/locale} or \texttt{TEXMFLOCALEDIR.cweb=$TEXMFMAIN/locale}.

(Include files 3*) $\equiv$
\begin{verbatim}
#include <locale.h>   \[ \text{LC\_MESSAGES, LC\_CTYPE} \]
#else
#define setlocale(A,B) ""
define bindtextdomain(A,B) ""
define textdomain(A) ""
#endif
\end{verbatim}

\[ 90^* \] (Set locale and bind language catalogs 90*) $\equiv$
\begin{verbatim}
setlocale(LC\_MESSAGES, setlocale(LC\_CTYPE,""));
texmf\_locale \leftarrow \texttt{kpse\_var\_expand("\$\{TEXMFLOCALEDIR\}"));
bindtextdomain("cweb", bindtextdomain("cweb-tl", bindtextdomain("web2c-help",
   strcmp(texmf\_locale,"") ? texmf\_locale : "/usr/share/locale"))));
   free(texmf\_locale);
textdomain("cweb");\[ \text{the majority of "strings" come from "plain CWEB"} \]
\end{verbatim}
This code is used in section 20*.
File lookup with KPATHSEA. The CTANGLE and CWEAVE programs from the original CWEB package use the compile-time default directory or the value of the environment variable CWEBINPUTS as an alternative place to be searched for files, if they could not be found in the current directory.

This version uses the KPATHSEA mechanism for searching files. The directories to be searched for come from three sources:

(a) a user-set environment variable CWEBINPUTS (overridden by CWEBINPUTS_cweb);
(b) a line in KPATHSEA configuration file texmf.cnf, e.g., CWEBINPUTS=$TEXMFDOTDIR:$TEXMF/texmf/cweb/ or CWEBINPUTS_cweb=$TEXMFDOTDIR:$TEXMF/texmf/cweb/;
(c) compile-time default directories (specified in texmf.in), i.e., $TEXMFDOTDIR:$TEXMF/texmf/cweb/.

We set kpse_program_name to ‘cweb’. This means if the variable CWEBINPUTS_cweb is present in texmf.cnf (or CWEBINPUTS_cweb in the environment) its value will be used as the search path for filenames. This allows different flavors of CWEB to have different search paths.

When the files you expect are not found, the thing to do is to enable KPATHSEA runtime debugging by assigning to the kpathsea_debug variable a small number via the ‘−d’ option. The meaning of this number is shown below. To set more than one debugging option, simply sum the corresponding numbers.

1 report ‘stat’ calls
2 report lookups in all hash tables
4 report file openings and closings
8 report path information
16 report directory list
32 report on each file search
64 report values of variables being looked up

Debugging output is always written to stderr, and begins with the string ‘kdebug:’.
System dependent changes. The most volatile stuff comes at the very end.

Modules for dealing with help messages and version info.

(Include files 3*) +≡
#define CWEWEB
#include "help.h"   ➢ CTANGLEHELP, CWEAVEHELP, CTWILLHELP <

(Display help message and exit 95*) ≡
cb_usagehelp(program ≡ ctangle ? CTANGLEHELP : program ≡ cweave ? CWEAVEHELP : CTWILLHELP, Λ);
This code is used in section 80*.

Special variants from Web2c’s 'lib/usage.c', adapted for i18n/t10n. We simply filter the strings through the catalogs (if available).

(Preamble of procedures 7*) +≡
static void cb_usage(const_string str);
static void cb_usagehelp(const_string *message, const_string bug_email);

static void cb_usage(const_string str)
{
  textdomain("cweb-1t"); fprintf(stderr, ("%s:\nNeed one to three file arguments.\n", str);
  fprintf(stderr, ("Try --help for more information.\n", str); textdomain("cweb");
  history ← fatal_message; exit(wrap_up());
}

static void cb_usagehelp(const_string *message, const_string bug_email)
{
  if (!bug_email) bug_email ← "tex-k@tug.org";
  textdomain("web2c-help");
  while (*message) {
    printf("%s\n", strcmp("", *message) ? *(message) : *message); ++message;
  }
  textdomain("cweb-1t"); printf(("\nEmail bug reports to %s.\n") , bug_email);
  textdomain("cweb"); history ← spotless; exit(wrap_up());
}

The version information will not be translated, it uses a generic text template in English.

(Display version information and exit 98*) ≡
printversionandexit(cb_banner,
  program ≡ ctwill ? "Donald E. Knuth": "Silvio Levy and Donald E. Knuth", Λ,
  "Contemporary development on https://github.com/ascherer/cweb.\n");
This code is used in section 80*.

But the “banner” is, at least the first part.

void cb_show_banner(void)
{
  assert(cb_banner[0] ≠ ’\0’); textdomain("cweb-1t"); printf("%s\n", (cb_banner), versionstring);
  textdomain("cweb");
}
100\* Index.
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, 23, 25, 29, 30, 32, 34, 36, 38, 39, 41, 42, 51, 57, 58, 60, 62, 67, 68, 69, 71, 73, 74, 75, 77, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100.

--help: 80* 89* 95* 97*
--quiet: 80*
--verbose: 80*
--version: 80* 98*
:: 3*
_DEV_NULL: 75*
ac: 13* 73*
active_file: 14* 83*
add_section_name: 55, 57* 61.
Ambiguous prefix ...: 60*
and_and: 4*
arg: 13* 73* 75*
argy: 13* 73* 75* 77* 78* 79* 80* 82* 92*
ASCII code dependencies: 4* 21.
assert: 99*
av: 13* 73*
bad_extension: 62* 63.
bindtextdomain: 87* 89* 90*
bool: 3*
boolean: 2* 6* 7* 8* 10* 13* 23* 24* 25* 37*, 38* 73* 75* 91*
buffer: 5* 22* 23* 26* 29* 30* 31* 32* 34* 35* 38* 40* 41* 42* 48* 66* 67* buffer_end: 5* 22* 23*
bug_email: 96* 97*
byte_mem: 2* 43* 44* 45.
byte_mem_end: 2* 43* 44* 45* 51* 57* 58* byte_nbr: 2* 44* 45* 51* 57* 58* byte_start: 2* 43* 44* 45* 51* 52* 53* 54* 57* 58* 63* c: 23* 57* 59* 63.
C_file: 13* 14* 73* 83* 84* 88* C_file_name: 13* 73* 77* 79* 84* C_print: 14* C_puts: 14*
Cannot open change file: 36*.
Cannot open input file: 36*.
Cannot open output file: 84*.
cbBanner: 87* 98* 99* cb_showBanner: 15* 87* 99* cb_usage: 81* 96* 97* cb_usageHelp: 95* 96* 97*
Change file ended...: 30* 32* 41*.
Change file entry did not match: 42*.
change_buffer: 26* 27* 31* 32* 40* 42* change_depth: 6* 25* 34* 35* 38* 40* 42* 67* change_file: 6* 25* 26* 29* 30* 32* 36* 37* 41* 67* change_file_name: 6* 25* 36* 75* 78* change_limit: 26* 27* 31* 32* 40* 42* change_line: 6* 25* 29* 30* 32* 35* 41* 67* change_pending: 8* 32* 37* 41* changed_section: 8* 32* 37* 41* changing: 6* 25* 26* 27* 32* 35* 38* 40* 41* 42* 67* check_change: 32* 33* 40* check_complete: 7* 42* check_file: 6* 14* 73* 83* 88* check_file_name: 6* 73* 84* 88* colon_colon: 4* common_init: 15* 20* 74* confusion: 11* 72* const_string: 91* 96* 97* ctangle: 2* 81* 84* 95* CTANGLEHELP: 94* 95* ctwill: 2* 98* C_TWILLHELP: 94* 95* cur_file: 6* 26* 32* 37* 39* 40* 67* cur_file_name: 6* 39* 67* cur_file_name_end: 39* cur_line: 6* 32* 35* 39* 40* 67* cweed: 2* 81* 95* CWAVEHELP: 94* 95* cweb: 2* 18* CWEB: 94*
CWEB file ended...: 32* cweb-\*\*\*: 89*, 97*, 99* cweb-\*: 89*, 97*, 99* CWEBINPUTS: 39, 91, 92*
dest: 53.
DEV_NULL: 75*
DEVICE_SEPARATOR: 75* 87*
DIR_SEPARATOR: 75* 87*
dotdotdot: 4*
dotpos: 75* 77* 78* 79* 80* 84* dummy: 9*
max_include_depth: 6* 25* 38*
max_names: 16* 43.
max_sections: 16* 37.
max_texts: 16*
max_lks: 16*
message: 96* 97*.
minus_gt: 4*.
minus_gt_ast: 4*.
minus_minus: 4*
Missing \0x...: 29*
n: 32*.
name: 91*.
n_name_dir: 9* 43, 44, 45, 52, 53, 57, 58* 60* 63.
n_name_dir_end: 9* 43, 44, 51, 57, 58*.
name_info: 9* 43.
n_name_len: 57* 58* 59, 60* 62*.
n_name_pointer: 9* 10* 43, 44, 46, 48, 52, 53, 54, 55, 57, 58, 59, 63, 64.
n_name_pos: 73* 77*.
n_name_ptr: 9* 44, 45, 48, 50, 51, 57, 58*.
names_match: 10* 46, 50.
New name extends...: 62*.
New name is a prefix...: 62*.
n_new_line: 14* 68*.
non_eq: 4*.
or_or: 4*
overflow: 12* 51* 57, 58* 70, 71*.
p: 43, 48, 52, 53, 54, 57, 58* 59.
par: 57* 59, 60* 61.
PATH_SEPARATOR: 87*.
period_ast: 4*.
pfirst: 63*.
phase: 2* 19*.
plus_plus: 4*.
prefix: 55, 56, 62* 63*.
prefix_length: 52, 54, 57, 60* 62*.
prime_the_change_buffer: 27, 28, 35, 41*.
print_id: 9*.
print_prefix_name: 10* 54, 60* 62*.
print_section_name: 10* 52, 62*.
print_stats: 15* 68*.
print_where: 8* 32* 37, 39* 40, 41*.
printf: 3* 34* 66, 67* 71* 97* 99*.
printversionandexit: 98*.
program: 2* 18* 81*, 84*, 95* 98*.
ptrdiff_t: 2*.
putc: 14*.
putchar: 14* 67*.
puts: 69*.
putchar: 14*.
q: 52, 53, 58* 59, 63*.
r: 59, 63*.
remove: 88*.
reset_input: 7* 35*.
restart: 38* 39*.
RETURN_ERROR: 68*.
RETURN_FAIL: 68*.
RETURN_OK: 68*.
RETURN_WARN: 68*.
Rlink: 9*.
rlink: 9* 43, 57* 60*.
root: 9* 45, 57* 59*.
s: 52, 53, 54, 57* 58* 63, 66, 70, 75*.
scan_args: 75*, 76, 84* 87*.
scn_file: 13* 14* 73* 83*.
scn_file_name: 13* 73* 77*, 79*.
Section name incompatible...: 62*.
section_count: 8* 32* 37, 41*.
section_lookup: 10* 59*.
section_name_cmp: 62*, 63, 64*.
section_text: 4* 21*.
section_text_end: 4* 21*.
SEPARATORS: 87*.
separators: 87*.
set_prefix_length: 52, 57* 62*.
ssetlocale: 89* 90*.
show_banner: 13* 80*.
show_happiness: 13* 69* 80*.
show_progress: 13* 68* 80*.
show_stats: 13* 68*, 74*.
sixteen_bits: 2* 8* 37*.
Sorry, capacity exceeded: 71*.
spaceless: 11* 65, 69* 97*.
sprintf_section_name: 10* 53*.
sprintf: 77*, 78, 79*.
ss: 52, 53, 63*.
sscanf: 80*.
stack_size: 16*.
stderr: 93*, 97*.
stdout: 14*, 60* 62*.
str: 96*, 97*.
strconv: 84*.
strcmp: 3* 36* 39* 55, 56, 78, 80* 90* 97*.
strcpy: 36* 39* 75* 77* 78, 79, 84*.
string: 87* 91*.
strlen: 3* 36* 39* 88*.
strncpy: 26, 36* 39*.
strcpy: 31, 42*, 51*, 53, 57*, 58* 75*.
strstr: 84*.
system dependencies: 21, 23*, 36* 39* 67*, 68* 75*, 83*.
t: 48, 70, 71*.
temporary_output: 13* 74*.
term_write: 9* 14*, 52, 54*.
tex_file: 13* 14*, 73*, 83*, 84*, 88*.
tex_file_name:    13*, 73*, 77*, 79*, 84*
texmf_locale:    87*, 90*
TEXMFLOCALEDIR:  89*
text_info:       43.
textdomain:      89*, 90*, 97*, 99*
This can’t happen: 11*
tolower:        29*, 32*, 41*
Too many nested includes: 38*
too_long:       39*
                 63, 74*, 77*, 78, 79, 80*, 91*
uint16_t:        2*, 3*
uint8_t:        2*, 3*
ungetc:         23*
update_terminal: 14*, 66.
Usage:          81*
use_language:   13*, 80*, 86*
versionstring:  91*, 99*
web_file:       6*, 25*, 36*, 42*
web_file_name:  6*, 36*, 77*
web_strcmp:     55, 56, 60*, 63.
web2c−help.mo:  89*, 97*
Where is the match...: 34*, 41*
wrap_up:        12*, 68*, 70, 97*
xisalpha:       5*
xisdigit:       5*
xislower:       5*
xisspace:       5*, 32*
xisupper:      5*, 29*, 32*, 41*
xisxdigit:     5*
xmem:          43.
xref:          43.
xyz_code:       32*, 34*
(Common code for **CWEAVE** and **CTANGLE** 2*, 4*, 5*, 6*, 8*, 9*, 11*, 13*, 14*)  Used in section 1*.

(Complain about argument length 82*)  Used in sections 77*, 78, and 79.

(Compute the hash code h 49)  Used in section 48.

(Compute the name location p 50)  Used in section 48.

(Display help message and **exit** 95*)  Used in section 80*.

(Display version information and **exit** 98*)  Used in section 80*.

(Enter a new name into the table at position p 51*)  Used in section 48.

(Global variables 18*, 19, 21, 22, 25*, 26, 37, 43, 44, 46, 65, 73*, 83*, 86*, 87*)  Used in section 1*.

(Handle flag argument 80*)  Used in section 75*.

(If no match found, add new name to tree 61)  Used in section 59.

(If one match found, check for compatibility and return match 62*)  Used in section 59.

(If the current line starts with @y, report any discrepancies and **return** 34*)  Used in section 32*.

(Include files 3*, 89*, 91*, 94*)  Used in section 1*.

(Initialize pointers 45, 47)  Used in section 20*.

(Look for matches for new name among shortest prefixes, complaining if more than one is found 60*)  Used in section 59.

(Make **change_file_name** from **fname** 78)  Used in section 75*.

(Make **web_file_name**, **tex_file_name**, and **C_file_name** 77*)  Used in section 75*.

(Move **buffer** and **limit** to **change_buffer** and **change_limit** 31)  Used in sections 27 and 32*.

(Open input files 36*)  Used in section 35.

(Override **tex_file_name** and **C_file_name** 79)  Used in section 75*.

(Predesclaration of procedures 7*, 10*, 12*, 15*, 24, 28, 33, 55, 64, 76, 96*)  Used in section 1*.

(Print error location based on input buffer 67*)  Used in section 66.

(Print the job **history** 69)  Used in section 68*.

(Print usage error message and quit 81*)  Used in sections 75* and 80*.

(Read from **change_file** and maybe turn off changing 41*)  Used in section 38*.

(Read from **cur_file** and maybe turn on changing 40)  Used in section 38*.

(Remove the temporary file if not already done 88)  Used in section 68*.

(Scan arguments and open output files 84*)  Used in section 20*.

(Set locale and bind language catalogs 90)  Used in section 20*.

(Set the default options common to **CTANGLE** and **CWEAVE** 74*)  Used in section 20*.

(Set up **PROGNAME** feature and initialize the search path mechanism 92*)  Used in section 20*.

(Skip over comment lines in the change file; **return** if end of file 29*)  Used in section 27.

(Skip to the next nonblank line; **return** if end of file 30*)  Used in section 27.

(Try to open include file, abort push if unsuccessful, go to **restart** 39*)  Used in section 38*.