May 6, 2022 at 15:14

1. Introduction. This program is a simple filter that inputs TEX or `CWEB` files and outputs its best guess at the "words" they contain. The word list can then be passed to a spelling check routine such as `wordtest`.

If this program is invoked with the name '`excweb`', it will apply special rules based on the syntax of `CWEB` files. Otherwise it will use only the TEX conventions. (Note that UNIX's `ln` command allows a program to be invoked with more than one name although it appears only once in the computer's memory.)

The TEX conventions adopted here say that words are what remain after you remove nonletters, control sequences, comments triggered by `%` marks, and material enclosed within `$...$` or `$$...$$`. However, an apostrophe within a word will be retained. The plain TEX control sequences for accented characters and special text characters, namely

```
\'     \`     \^     \"     \~     \=     \.     \u     \v
\H     \t     \c     \d     \b     \oe    \OE    \ae    \AE
\aa    \AA    \o     \O     \l     \L     \ss    \i     \j
```

will also be retained, so that users can treat them as parts of words. A blank space following any of the alphabetic control sequences in this list will be carried along too. If any of these control sequences is followed by `{`, everything up to the next `}` will also be retained. Thus, for example, the construction '`m\={\i}n\u␣us`' will be considered a single word, in spite of the control sequences and the space between the two u's. Discretionary hyphens '`\-`' are treated in the same way as accents.

The `CWEB` conventions are essentially the same as the TEX conventions, in the TEX parts of a `CWEB` file. The C parts of the file are blanked out.

No attempt is made to reach a high level of artificial intelligence, which would be able to truly understand the input file. Tricky users can confuse us. But we claim that devious tricks are their problem, not ours.

2. So here goes. The main idea is to keep a one-character lookahead buffer, called *c*, which is set to zero when the character has been processed. A giant switch to various cases, depending on the value of *c*, keeps everything moving.

If you don't like **goto** statements, don't read this. (And don't read any other programs that simulate finite-state automata.)

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
  extern void exit();     /* system routine that terminates execution */
  ⟨Global variables 3⟩
  ⟨Procedures 6⟩
  int main(argc, argv)
       int argc;     /* the number of arguments (should be 1, but this isn't checked) */
       char *argv[];     /* the arguments (*argv is the program name) */
  {
    ⟨Local variables 4⟩;
    if (strlen(*argv) ≥ 6 ∧ strcmp(*argv + strlen(*argv) − 6, "excweb") ≡ 0) {
      web = 1;
      ⟨Adjust tables for CWEB mode 15⟩;
    }
    else web = 0;
    comment = skipping = c = 0;
  main_cycle:
    if (c) goto big_switch;
  restart: c = get();
  big_switch:
    switch (c) {
      ⟨Special cases of the giant switch where we don't just discard c 5⟩
    case EOF: exit(0);
    default: goto restart;
    }
    ⟨Labeled code segments, which exit by explicit goto 8⟩;
  }
```

3. ⟨Global variables 3⟩ ≡

```
  int c;     /* one-character look-see buffer */
```

See also sections 14 and 17.

This code is used in section 2.

4. ⟨Local variables 4⟩ ≡

```
  int web;     /* are we looking for CWEB constructs? */
  int comment;     /* are we inside a C comment in a CWEB document? */
  int skipping;     /* are we skipping C code in a CWEB document? */
  int save_skipping;     /* value of skipping outside current C mode */
  register int cc;     /* temporary buffer */
```

This code is used in section 2.

5. Simple cases. Let's do some of the easiest things first, in order to get the hang of this program. Several special characters will cause us to ignore everything until the first appearance of something else.

```
#define discard_to(x)
        { while (get() ≠ x) ; }
#define discard_to_dol
        { for (cc = c, c = get(); c ≠ '$' ∨ cc ≡ '\\'; cc = c, c = get())
            if (cc ≡ '\\' ∧ c ≡ cc) c = '\0'; }
```

⟨ Special cases of the giant switch where we don't just discard c 5 ⟩ ≡

```
case '%': discard_to('\n'); goto restart;
case '$': c = getchar();
  if (c ≠ '$') discard_to_dol
  else {      /* after $$ we discard everything to the next $$ */
    do discard_to_dol while (getchar() ≠ '$');
  }
  goto restart;
```

See also sections 7, 11, and 16.

This code is used in section 2.

6. The '*get*' procedure in the code above is like C's standard '*getchar*', except that it immediately terminates execution at the end of the input file. Otherwise malformed input files could lead to infinite loops.

⟨ Procedures 6 ⟩ ≡

```
  int get()
  { register int x;
    x = getchar();
    if (x ≡ EOF) exit(0);
    return x;
  }
```

See also section 12.

This code is used in section 2.

7. More complex behavior is handled by jumping out of the **switch** statement to one of the routines following it. None of the cases say **break**, so the code following the switch statement is accessible only via **goto**.

⟨ Special cases of the giant switch where we don't just discard c 5 ⟩ +≡

```
case 'a': case 'A': case 'b': case 'B': case 'c': case 'C': case 'd': case 'D': case 'e':
  case 'E': case 'f': case 'F': case 'g': case 'G': case 'h': case 'H': case 'i': case 'I':
  case 'j': case 'J': case 'k': case 'K': case 'l': case 'L': case 'm': case 'M': case 'n':
  case 'N': case 'o': case 'O': case 'p': case 'P': case 'q': case 'Q': case 'r': case 'R':
  case 's': case 'S': case 't': case 'T': case 'u': case 'U': case 'v': case 'V': case 'w':
  case 'W': case 'x': case 'X': case 'y': case 'Y': case 'z': case 'Z': goto out_word;
```

8. When letters appear in *stdin*, we pass them immediately through to *stdout* with little further ado. An apostrophe is rejected unless it is immediately followed by a letter.

```
⟨ Labeled code segments, which exit by explicit goto 8 ⟩ ≡
out_word: putchar(c);
continue_word: c = getchar();
checkout_word:
  if (isalpha(c)) goto out_word;
  if (c ≡ '\'') {
    c = getchar();
    if (isalpha(c)) {
      putchar('\''); goto out_word;
    }
    goto end_word;
  }
  if (c ≡ '\\' ∧ controlseq()) goto control_seq_in_word;
end_word: putchar('\n');
  goto main_cycle;
```

See also sections 10, 18, and 19.

This code is used in section 2.

9. Control sequences. The *controlseq*( ) function is the only delicate part of this program. After a backslash has been scanned, *controlseq* looks to see if the next characters define one of the special plain TEX macros listed above. If so, the control sequence and its immediately following argument (if any) are output and *controlseq* returns a nonzero value. If not, nothing is output and *controlseq* returns zero. In both cases the value of c will be nonzero if and only if *controlseq* has had to look ahead at a character it decided not to process.

10. ⟨ Labeled code segments, which exit by explicit **goto** 8 ⟩ +≡

```
control_seq_in_word:
  if (¬c) goto continue_word;
  goto checkout_word;
```

11. ⟨ Special cases of the giant switch where we don't just discard c 5 ⟩ +≡

```
case '\\':
  if (controlseq( )) goto control_seq_in_word;
  goto main_cycle;
```

12. ⟨ Procedures 6 ⟩ +≡

```
  int controlseq( )
  {
    int l;    /* number of letters in the control sequence */
    char a, b;    /* the first two characters after '\' */
    l = 0;
    a = c = getchar( );
    while (isalpha(c)) {
      l++;
      c = getchar( );
      if (l ≡ 1) b = c;
    }
    if (l ≡ 0) c = getchar( );
    ⟨ Check for special plain TEX control sequences; output them and return 1 if found 13 ⟩;
    return 0;
  }
```

13. **#define** *pair*(x, y) $(a \equiv x \wedge b \equiv y)$

⟨ Check for special plain TEX control sequences; output them and **return** 1 if found 13 ⟩ ≡

```
if ((a ≥ '"' ∧ a ≤ '~' ∧ ptab[a − '"'] ≡ l) ∨ (l ≡ 2 ∧ (pair('a', 'e') ∨ pair('A', 'E')
      ∨ pair('o', 'e') ∨ pair('O', 'E')
      ∨ pair('a', 'a') ∨ pair('A', 'A') ∨ pair('s', 's')))) {
  putchar('\\');
  putchar(a);
  if (l ≡ 2) putchar(b);
  if (l ∧ c ≡ '␣') {
    putchar('␣');    /* optional space after alphabetic control sequence */
    c = getchar();
  }
  if (c ≡ '{') {
    do {
      putchar(c);
      c = get();
    } while (c ≠ '}');    /* optional argument after special control sequence */
    putchar(c);
    c = 0;
  }
  return 1;
}
```

This code is used in section 12.

14. The *ptab* entries for nonletters are 0 when the control sequence is special, otherwise 1; the conventions for letters are reversed.

⟨ Global variables 3 ⟩ +≡

```
char ptab[] = {0, 1, 1, 1, 1, 0,    /* \" and \' */
1, 1, 1, 1, 1, 0, 0, 1,    /* \- and \. */
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,    /* \= */
1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1,    /* \H, \L, \O */
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1,    /* \^ */
0, 0, 1, 1, 1, 0, 0, 0,    /* \`, \b, \c, \d */
0, 1, 1, 0, 1, 0, 0, 1,    /* \i, \j, \l, \o */
0, 0, 0, 0, 1, 1, 1, 0,    /* \t, \u, \v */
0, 0, 0, 1, 1, 1, 0};    /* \~ */
```

15. In `CWEB` the TEX control sequence '`\.`' denotes the typewriter font used for strings, not the dot-over accent. We must modify *ptab* to reflect this unfortunate (but too-late-too-change) design decision.

⟨ Adjust tables for `CWEB` mode 15 ⟩ ≡

```
ptab[12] = 1;
```

This code is used in section 2.

16. CWEB considerations. We're finished now with all that would be needed if we only wanted to handle TEX. For `CWEB` a bit more should be done.

The `CWEB` escape character is `@`, and the character following it tells us what mode we should enter. In TEX mode we should not only do what we normally do for TEX files, we should also ignore material delimited by `|...|`, being careful to recognize when `'|'` is part of a string (as it just was). And we should stop TEX mode if we scan `*/` while in a C comment.

```
⟨Special cases of the giant switch where we don't just discard c 5⟩ +≡
case '@':
  if (web) goto do_web;
  goto restart;
case '|':
  if (web > 1) {
    save_skipping = skipping;
    goto skip_C_prime;
  }
  goto restart;
case '*':
  if (¬comment) goto restart;
  c = getchar();
  if (c ≡ '/') {
    comment = 0;
    goto skip_C;
  }
  goto big_switch;
```

17. The characters that follow `@` in a `CWEB` file can be classified into a few types that have distinct implications for `excweb`.

```
#define nop 0   /* control code that doesn't matter to us */
#define start_section 1   /* control code that begins a CWEB section */
#define start_C 2   /* control code that begins C code */
#define start_name 3   /* control code that begins a section name */
#define start_index 4   /* control code for CWEB index entry */
#define start_insert 5   /* control code for C material ended by '@>' */
#define end_item 6   /* '@>' */
#define ignore_line 7   /* '@i' or '@l' */
```

```
⟨Global variables 3⟩ +≡
  char wtab[] = {start_section, nop, nop, nop, nop, nop, nop, nop,   /* ␣!"#$%&' */
  start_name, nop, start_section, nop, nop, nop, start_index, nop,   /* ()*+,-./ */
  nop, nop, nop, nop, nop, nop, nop, nop,   /* 01234567 */
  nop, nop, start_index, nop, start_name, start_insert, end_item, nop,   /* 89:;<=>? */
  nop, nop, nop, start_C, start_C, nop, start_C, nop,   /* @ABCDEFG */
  nop, ignore_line, nop, nop, ignore_line, nop, nop, nop,   /* HIJKLMNO */
  start_C, start_insert, nop, start_C, start_insert, nop, nop, nop,   /* PQRSTUVW */
  nop, nop, nop, nop, nop, nop, start_index, nop,   /* XYZ[\]^_ */
  nop, nop, nop, start_C, start_C, nop, start_C, nop,   /* `abcdefg */
  nop, ignore_line, nop, nop, ignore_line, nop, nop, nop,   /* hijklmno */
  start_C, start_insert, nop, start_C, start_insert};   /* pqrst */
```

18. We do not leave TEX mode until the first WEB section has begun.

```
⟨ Labeled code segments, which exit by explicit goto 8 ⟩ +≡
do_web: c = getchar();
  if (c < '␣' ∨ c > 't') goto restart;
  switch (wtab[c − '␣']) {
  case nop: case start_index: goto restart;
  case start_section: web++;      /* out of "limbo" */
    comment = skipping = 0; goto restart;
  case start_C:
    if (web > 1) goto skip_C;
    goto restart;
  case start_name: case start_insert:
    if (web > 1) skipping = 1;
    goto restart;
  case end_item:
    if (skipping) goto skip_C;
    goto restart;
  case ignore_line: discard_to('\n');
    goto restart;
  }
```

19. The final piece of program we need is a sub-automaton to pass over the C parts of a `CWEB` document. The main subtlety here is that we don't want to get out of synch while scanning over a supposed string constant or verbatim insert.

```
⟨Labeled code segments, which exit by explicit goto 8⟩ +≡
skip_C: save_skipping = 2;
skip_C_prime: skipping = 1;
  while (1) {
    c = get();
  C_switch:
    switch (c) {
    case '/': c = get();
      if (c ≠ '*') goto C_switch;
      comment = 1;    /* fall through to the next case, returning to TeX mode */
    case '|':
      if (save_skipping ≡ 2) continue;    /* '|' as C operator */
      skipping = save_skipping; goto restart;    /* '|' as CWEB delimiter */
    case '@': c = getchar();
    inner_switch:
      if (c < '␣' ∨ c > 't') continue;
      switch (wtab[c − '␣']) {
      case nop: case start_C: case end_item: continue;
      case start_section: web++;
        comment = skipping = 0; goto restart;
      case start_name: case start_index: goto restart;
      case start_insert: do discard_to('@') while ((c = getchar()) ≡ '@');
        goto inner_switch;    /* now c should equal '>' */
      case ignore_line: discard_to('\n');
        continue;
      }
    case '\'': case '"':
      while ((cc = get()) ≠ c ∧ cc ≠ '\n')
        if (cc ≡ '\\') getchar();
        else if (cc ≡ '@') {
          cc = getchar();
          if (cc ≠ '@') {    /* syntax error, we try to recover */
            c = cc;
            goto inner_switch;
          }
        }
      ;
      continue;    /* CWEB strings do not extend past one line */
    default: continue;
    }
  }
```

20. Index.

⟨ Adjust tables for `CWEB` mode 15 ⟩ Used in section 2.
⟨ Check for special plain TEX control sequences; output them and **return** 1 if found 13 ⟩ Used in section 12.
⟨ Global variables 3, 14, 17 ⟩ Used in section 2.
⟨ Labeled code segments, which exit by explicit **goto** 8, 10, 18, 19 ⟩ Used in section 2.
⟨ Local variables 4 ⟩ Used in section 2.
⟨ Procedures 6, 12 ⟩ Used in section 2.
⟨ Special cases of the giant switch where we don't just discard c 5, 7, 11, 16 ⟩ Used in section 2.

EXTEX