REFSORT

June 11, 2023 at 13:14

1*. Introduction. This short program sorts the mini-indexes of listings prepared by `CTWILL`.

More precisely, suppose you have said `ctwill foo.w`, getting a file `foo.tex`, and that you've then said `tex foo.tex`, getting files `foo.dvi` and `foo.ref`. If you're happy with `foo.dvi` except for the alphabetic order of the mini-indexes, you can then say

```
refsort <foo.ref >foo.sref
```

after which `tex foo` will produce `foo.dvi` again, this time with the mini-indexes in order.

Still more precisely, this program reads from standard input a file consisting of groups of unsorted lines and writes to standard output a file consisting of groups of sorted lines. Each input group begins with an identification line whose first character is `!`; the remaining characters are a page number. The other lines in the group all have the form

$$\texttt{+}\sqcup\alpha\sqcup\texttt{\textbackslash?\{}\kappa\texttt{\}}\omega$$

where α is a string containing no spaces, `?` is a single character, κ is a string of letters, digits, and `\_`'s, and ω is an arbitrary string. The output groups contain the same lines without the initial `+␣`, sorted alphabetically with respect to the κ fields, followed by a closing line that says '`\donewithpage`' followed by the page number copied from the original identification line.

Exception: In the case of a "custom" identifier, `\?{`κ`}` takes the alternative form `$\`κ`␣$` instead.

We define limits on the number and size of mini-index entries that should be plenty big enough.

#define *max_key* 50 ▷ greater than the length of the longest identifier ◁
#define *max_size* 120 ▷ greater than the length of the longest mini-index entry ◁
#define *max_items* 300 ▷ the maximum number of items in a single mini-index ◁

2*. Here's the layout of the C program:

```
#define abort(c, m)
        {
          fprintf(stderr, "%s!\n%s", m, buf);  return c;
        }
#include <stdio.h>
#include <string.h>
#include <ctype.h>
  typedef struct {
    char key[max_key];
    char entry[max_size];
  } item;
  item items[max_items];      ▷ all items of current group ◁
  item *sorted[max_items];      ▷ pointers to items in alphabetic order ◁
  char cur_page[10];      ▷ page number, as a string ◁
  char buf[max_size];      ▷ current line of input ◁
  char *input_status;      ▷ Λ if end of input reached, else buf ◁
  int main()
  {
    char *p, *q;
    int n;      ▷ current number of items ◁
    item *x, **y;
    input_status ← fgets(buf, max_size, stdin);
    while (input_status) {
      ⟨ Check that buf contains a valid page-number line 3 ⟩;
      ⟨ Read and sort additional lines, until buf terminates a group 4 ⟩;
      ⟨ Output the current group 5* ⟩;
    }
    return 0;      ▷ normal exit ◁
  }
```

5*. ⟨ Output the current group 5* ⟩ ≡

```
  {
    for (y ← sorted; y < sorted + n; y++) printf("%s\n", (*y)→entry);
    printf("\\donewithpage%s\n", cur_page);
  }
```

This code is used in section 2*.

9*. A bugfix. The program specification had a subtle bug: There are cases where α includes spaces that should be removed in the output.

These cases occur when a space occurs after an odd number of doublequote characters. Ergo, the following routine replaced a simpler original loop.

```
⟨ Scan past $\alpha$ 9* ⟩ ≡
  {
    int toggle ← 0;
    for (p ← buf + 2; (*p ≠ '␣' ∨ toggle) ∧ *p; p++)
      if (*p ≡ '"') toggle ⊕= 1;
  }
```

This code is used in section 6.

10*. A corresponding change to the copying loop is also needed.

```
⟨ Copy the buffer to x→entry 10* ⟩ ≡
  {
    int toggle ← 0;
    for (p ← buf + 2, q ← x→entry; (*p ≠ '␣' ∨ toggle) ∧ *p; p++) {
      if (*p ≡ '"') toggle ⊕= 1;
      if (*p ≠ '␣') *q++ ← *p;
    }
    for ( ; *p; p++) *q++ ← *p;
  }
```

This code is used in section 6.

11* Index.

The following sections were changed by the change file: 1, 2, 5, 9, 10, 11.