REFSORT

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Sorting</td>
<td>6</td>
</tr>
<tr>
<td>A bugfix</td>
<td>9</td>
</tr>
<tr>
<td>Index</td>
<td>11</td>
</tr>
</tbody>
</table>
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

```
+α κ ω
```

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 ◀
```
Here's the layout of the C program:

```c
#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("\ndonewithpage%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 $\alpha$ 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.

\[
\text{(Scan past $\alpha$ 9*)} \equiv \\
\{ \text{int} \ toggle \leftarrow 0; \\
\quad \text{for} \ (p \leftarrow \text{buf} + 2; (*p \neq \text{' '}) \lor toggle) \land *p; \ p++) \\
\quad \quad \text{if} \ (*p \equiv \text{'"'}) \ toggle \ominus = 1; \\
\} \\
\]

This code is used in section 6.

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

\[
\text{(Copy the buffer to $x$-entry 10*)} \equiv \\
\{ \text{int} \ toggle \leftarrow 0; \\
\quad \text{for} \ (p \leftarrow \text{buf} + 2, q \leftarrow \text{x-entry}; (*p \neq \text{' '}) \lor toggle) \land *p; \ p++) \{ \\
\quad \quad \text{if} \ (*p \equiv \text{'"'}) \ toggle \ominus = 1; \\
\quad \quad \text{if} \ (*p \neq \text{' '}) \ *q++ \leftarrow *p; \\
\quad \} \\
\quad \text{for} \ (; *p; \ p++) \ *q++ \leftarrow *p; \\
\} \\
\]

This code is used in section 6.
11* Index.
The following sections were changed by the change file: 1, 2, 5, 9, 10, 11.

abort: 2*, 3, 4, 6, 7.
buf: 2*, 3, 4, 6, 9*, 10*
cur_page: 2*, 3, 5*
entry: 2*, 5*, 10*
fgets: 2*, 4.
fprintf: 2*
input_status: 2*, 4.
isupper: 6, 7.
item: 2*
items: 2*, 4.
key: 2*, 6, 7, 8.
main: 2*
max_items: 1*, 2*, 4.
max_key: 1*, 2*, 6.
max_size: 1*, 2*, 4, 6.
n: 2*
p: 2*
printf: 5*
q: 2*
sorted: 2*, 5*, 8.
stderr: 2*
stdin: 2*, 4.
strcmp: 8.
strlen: 3.
toggle: 9*, 10*
x: 2*
y: 2*
(Check that \textit{buf} contains a valid page-number line 3) Used in section 2*.
(Copy the buffer to \textit{x-entry} 10*) Used in section 6.
(Copy \textit{buf} to item \textit{x} 6) Used in section 4.
(Output the current group 5*) Used in section 2*.
(Process a custom-formatted identifier 7) Used in section 6.
(Read and sort additional lines, until \textit{buf} terminates a group 4) Used in section 2*.
(Scan past \textit{α} 9*) Used in section 6.
(Sort the new item into its proper place 8) Used in section 4.