<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>
</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

```
+\alpha\kappa\omega
```

where $\alpha$ is a string containing no spaces, $\kappa$ is a single character, $\kappa$ is a string of letters, digits, and \_’s, and $\omega$ is an arbitrary string. The output groups contain the same lines without the initial `+`, sorted alphabetically with respect to the $\kappa$ 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, `$\kappa$\omega` takes the alternative form `$\kappa\omega$` instead.

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

```
#define max_key 30   // greater than the length of the longest identifier
#define max_size 100 // 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 "strings.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;  // \ if end of input reached, else buf <

main() {
   register char *p, *q;
   register int n;  // current number of items <
   register 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 <
   }
}
```

3. ⟨Check that buf contains a valid page-number line 3⟩ ≡
   if (*buf ≠ '!' ) abort(−1, "missing..!'!");
   if (strlen(buf + 1) > 11) abort(−2, "page..number..too..long");
   for (p ← buf + 1, q ← cur_page; *p ≠ '\n'; p++ ) *q++ ← *p;
   *q ← '\0';
This code is used in section 2.

4. ⟨Read and sort additional lines, until buf terminates a group 4⟩ ≡
   n ← 0;
   while (1) {
      input_status ← fgets(buf, max_size, stdin);
      if (input_status ≡ \ ∨ *buf ≠ '+') break;
      x ← &items[n];  ⟨Copy buf to item x 6⟩;
      ⟨Sort the new item into its proper place 8⟩;
      if (++n > max_items) abort(−11, "too..many..lines..in..group");
   }
This code is used in section 2.
5. \( \langle \text{Output the current group} 5 \rangle \equiv \)

\[
\begin{array}{l}
\text{register int } k; \\
\text{for (} y \leftarrow \text{sorted}; y < \text{sorted} + n; y++ \text{) printf("%s
", (*y)-entry); } \\
\text{printf("\\donewithpage\%s\n", cur_page); }
\end{array}
\]

This code is used in section 2.
6. Sorting. We convert the key to lowercase as we copy it, and we omit backslashes. We also convert \_ to \_. Then \_ will be alphabetically less than alphabetic letters, as desired.

(\text{Copy buf to item x}) \equiv 
\begin{align*}
\text{if } \& (\text{buf} + 1) \neq \textquoteleft ', \textquoteright & \text{ abort}(−3, "missing\_blank\_after\_\textquoteleft '\ textquoteright +"); \\
\text{Scan past } 9; \\
\text{if } (*p \neq '9) \text{ abort}(−4, "missing\_blank\_after\_alpha"); \\
\text{if } (*p + 1 \equiv '9) \langle \text{Process a custom-formatted identifier } \rangle \text{ else } \\
\{ \\
\text{if } (*p + 1 \neq '9) \text{ abort}(−5, "missing\_backslash"); \\
\text{if } (\neg(p + 2)) \text{ abort}(−6, "missing\_control\_code"); \\
\text{if } (*p + 3) \neq '9) \text{ abort}(−7, "missing\_left\_brace"); \\
\text{for } (p += 4, q ← x\_key; *p \neq '9) \wedge *p; p++) \{ \\
\text{if } (*p \neq '9) \langle \text{Process a custom-formatted identifier } \rangle \text{ else } \\
\{ \\
\text{if } (\text{isupper}(p)) \ *q++ ← *p + ('a' − 'A'); \\
\text{else if } (*p \equiv '_' \text{) } *q++ ← '_' \text{; \\
\text{else } *q++ ← *p; \\
\} \\
\text{if } (*p \neq '9) \text{ abort}(−8, "missing\_right\_brace"); \\
\}
\text{if } (q \geq &x\_key[\text{max\_key}]) \text{ abort}(−9, "key\_too\_long"); \\
*q ← '0'; \langle \text{Copy the buffer to x\_entry } \rangle \\
\text{if } (p \equiv &buf + max\_size - 1) \text{ abort}(−10, "entry\_too\_long"); \\
*(q - 1) ← '\0'; \\
\} \text{This code is used in section 4.}
\end{align*}

7. \langle \text{Process a custom-formatted identifier } \rangle \equiv 
\begin{align*}
\{ \\
\text{if } (*p + 2) \neq '9 \text{ abort}(−11, "missing\_custom\_backslash"); \\
\text{for } (p += 3, q ← x\_key; *p \neq '9) \wedge *p; p++) \{ \\
\text{if } (\text{isupper}(p)) \ *q++ ← *p + ('a' − 'A'); \\
\text{else } *q++ ← *p; \\
\} \\
\text{if } (*p \neq '9) \text{ abort}(−12, "missing\_custom\_space"); \\
\text{if } (*p + 1 \neq '9) \text{ abort}(−13, "missing\_custom\_dollarsign"); \\
\}
\text{This code is used in section 6.}
\end{align*}

8. \langle \text{Sort the new item into its proper place } \rangle \equiv 
\begin{align*}
\text{for } (y ← &\text{sorted}[n]; y > &\text{sorted}[0] \wedge \text{strcmp}((y - 1)\_key, x\_key) > 0; y -- ) \ *y ← *(y - 1); \\
*y ← x; \\
\} \text{This code is used in section 4.}
\end{align*}
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.

\[
\text{Scan past } \alpha \equiv \\
\begin{array}{l}
\text{register int } \text{toggle} \leftarrow 0; \\
\text{for } (p \leftarrow \text{buf} + 2; (*p \neq 'u' \lor \text{toggle}) \land *p; p++) \\
\quad \text{if } (*p \equiv '} \text{toggle } \oplus = 1; \\
\end{array}
\]

This code is used in section 6.

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

\[
\text{Copy the buffer to } x\text{-entry } \equiv \\
\begin{array}{l}
\text{register int } \text{toggle} \leftarrow 0; \\
\text{for } (p \leftarrow \text{buf} + 2, q \leftarrow x\text{-entry}; (*p \neq 'u' \lor \text{toggle}) \land *p; p++) \\
\quad \text{if } (*p \equiv '} \text{toggle } \oplus = 1; \\
\quad \text{if } (*p \neq 'u') *q++ \leftarrow *p; \\
\end{array}
\]

This code is used in section 6.

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.
k: 5.
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 $buf$ contains a valid page-number line 3) Used in section 2.
Copy the buffer to $x$-entry 10) Used in section 6.
Copy $buf$ to item $x$ 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 $buf$ terminates a group 4) Used in section 2.
Scan past $\alpha$ 9) Used in section 6.
Sort the new item into its proper place 8) Used in section 4.