Problem 3—Circular Buffer
written by Dave Powers

Watching Michelle Kwan skate those beautiful circles, I am reminded of a circular queue. A circular queue works the same as a regular queue: a last-in, first-out data structure. You write stuff and read stuff from opposite ends. A circular queue is one in which there is no physical beginning or ending of the data structure. You could (if you're not careful) fill the entire queue with stuff and then overwrite existing stuff when your write end comes in contact with the read end.

A circular queue is used as a communications buffer. The buffer has room for 54 characters, with positions numbered between 0 and 53. The read end and the write end both start at position 0. Writing never changes the read end. Reading never changes the write end.

INPUT SPECIFICATION. The input will contain multiple lines. Each line will contain a command. The command will either be an "R" followed by an unsigned decimal integer to read (and discard) from the buffer the specified number of characters, a "W" followed by a string (possibly empty) to write to the buffer, or a "0" which means the end of input. Any read command that requests more characters than are currently in the buffer will receive all available characters and will empty the buffer. Any write command that tries to write more characters than there is space available will be ignored in its entirety. There will be no spaces separating "R" or "W" from its argument. All lines are terminated by <**EOLN**>.

OUTPUT SPECIFICATION. For each command, output the next available position within the buffer for writing, a comma and a space, and how much free space is available. Terminate the line with <**EOLN**>. When the last command is read ("0"), output the current contents of the buffer followed by <**EOLN**>.

SAMPLE INPUT.

```
R20<EOLN>
WThis·is·my·name.<EOLN>
R5<EOLN>
0<EOLN>
<EOF>
```

SAMPLE OUTPUT.

```
0,·54<EOLN>
16,·38<EOLN>
16,·43<EOLN>
is·my·name.<EOLN>
<EOF>
```