Here's a sample run:

```
see ken run
#really fast
seekenrun
9 characters read
```

Apparently Ken runs so fast, he obliterates space itself—or at least the space characters in the input.

**Program Notes**

First, note the structure. The program reads the first input character before it reaches the loop. That way, the first character can be tested when the program reaches the loop statement. This is important, for the first character might be 

Because `textin1.cpp` uses an entry-condition loop, the program correctly skips the entire loop in that case. And because the variable `count` previously was set to zero, `count` has the correct value.

Suppose the first character read is not a 

Then, the program enters the loop, displays the character, increments the count, and reads the next character. This last step is vital. Without it, the loop repeatedly processes the first input character forever. With it, the program advances to the next character.

Note the loop design follows the guidelines mentioned earlier. The condition that terminates the loop is if the last character read is 

The condition is initialized by reading a character before the loop starts. The condition is updated by reading a new character at the end of the loop.

This all sounds reasonable. So why does the program omit the spaces on output? Blame `cin`. When reading type `char` values, just as when reading other basic types, `cin` skips over spaces and newlines. The spaces in the input are not echoed, and so they are not counted.

To further complicate things, the input to `cin` is buffered. That means the characters you type don't get sent to the program until you press Enter. This is why we were able to type characters after the 

After we pressed Enter, the whole sequence of characters was sent to the program, but the program quit processing the input after it reached the # character.