A: Nope. As part of C's UNIX heritage, it always regards the end of a line as being marked by a single line-feed character. (In UNIX text files, a single line-feed character—but no carriage return—appears at the end of each line.) The C library takes care of translating the user's keypress into a line-feed character. When a program reads from a file, the I/O library translates the file's end-of-line marker (whatever it may be) into a single line-feed character. The same transformations occur—in reverse—when output is written to the screen or to a file. (See Section 22.1 for details.)

Although these translations may seem confusing, they serve an important purpose: insulating programs from details that may vary from one operating system to another.

***Q: What's the purpose of the \? escape sequence? [p. 138]**

A: The \? escape is related to trigraph sequences, which begin with ??. If you should put ?? in a string, there's a possibility that the compiler will mistake it for the beginning of a trigraph. Replacing the second ? by \? fixes the problem.

trigraph sequences ➤25.3

Q: If `getchar` is faster, why would we ever want to use `scanf` to read individual characters? [p. 140]

A: Although it's not as fast as `getchar`, the `scanf` function is more flexible. As we saw previously, the `"%c"` format string causes `scanf` to read the next input character; `" %c"` causes it to read the next non-white-space character. Also, `scanf` is good at reading characters that are mixed in with other kinds of data. Let's say that our input data consists of an integer, then a single nonnumeric character, then another integer. By using the format string `"%d%c%d"`, we can get `scanf` to read all three items.

***Q: Under what circumstances do the integral promotions convert a character or short integer to `unsigned int`? [p. 143]**

A: The integral promotions yield an `unsigned int` if the `int` type isn't large enough to include all possible values of the original type. Since characters are usually eight bits long, they are almost always converted to `int`, which is guaranteed to be at least 16 bits long. Signed short integers can always be converted to `int` as well. Unsigned short integers are problematic. If short integers have the same length as ordinary integers (as they do on a 16-bit machine), then unsigned short integers will have to be converted to `unsigned int`, since the largest unsigned short integer (65,535 on a 16-bit machine) is larger than the largest `int` (32,767).

Q: Exactly what happens if I assign a value to a variable that's not large enough to hold it? [p. 146]

A: Roughly speaking, if the value is of an integral type and the variable is of an unsigned type, the extra bits are thrown away; if the variable has a signed type, the result is implementation-defined. Assigning a floating-point number to a variable—integer or floating—that's too small to hold it produces undefined behavior: anything can happen, including program termination.