decimal numbers are also useful for defining constants with extreme values, such as the values of the macros in the `<float.h>` header. These constants are easy to write in hex but difficult to write in decimal.

***Q: Why do we use `%lf` to read a `double` value but `%f` to print it? [p. 134]**

A: This is a tough question to answer. First, notice that `scanf` and `printf` are unusual functions in that they aren't restricted to a fixed number of arguments. We say that `scanf` and `printf` have variable-length argument lists. When functions with variable-length argument lists are called, the compiler arranges for `float` arguments to be converted automatically to type `double`. As a result, `printf` can't distinguish between `float` and `double` arguments. This explains why `%f` works for both `float` and `double` arguments in calls of `printf`.

variable-length argument lists ➤26.1

`scanf`, on the other hand, is passed a *pointer* to a variable. `%f` tells `scanf` to store a `float` value at the address passed to it, while `%lf` tells `scanf` to store a `double` value at that address. The distinction between `float` and `double` is crucial here. If given the wrong conversion specification, `scanf` will likely store the wrong number of bytes (not to mention the fact that the bit pattern for a `float` isn't the same as that for a `double`).

Q: What's the proper way to pronounce `char`? [p. 134]

A: There's no universally accepted pronunciation. Some people pronounce `char` in the same way as the first syllable of "character." Others say "char," as in

```
char broiled;
```

Q: When does it matter whether a character variable is signed or unsigned? [p. 136]

A: If we store only 7-bit characters in the variable, it doesn't matter, since the sign bit will be zero. If we plan to store 8-bit characters, however, we'll probably want the variable to have `unsigned char` type. Consider the following example:

```
ch = '\xdb';
```

If `ch` has been declared to have type `char`, the compiler may choose to treat it as a signed character (many compilers do). As long as `ch` is used only as a character, there won't be any problem. But if `ch` is ever used in a context that requires the compiler to convert its value to an integer, we're likely to have trouble: the resulting integer will be negative, since `ch`'s sign bit is 1.

Here's another situation: In some kinds of programs, it's customary to use `char` variables to store one-byte integers. If we're writing such a program, we'll have to decide whether each variable should be `signed char` or `unsigned char`, just as we must decide whether ordinary integer variables should have type `int` or `unsigned int`.

Q: I don't understand how the new-line character can be the ASCII line-feed character. When a user enters input and presses the Enter key, doesn't the program read this as a carriage-return character or a carriage return plus a line feed? [p. 137]