A: Some C libraries supply functions with names like `itoa` that convert numbers to strings. Using these functions isn't a great idea, though: they aren't part of the C standard and won't be portable. The best way to perform this kind of conversion is to call a function such as `sprintf` that writes formatted output into a string:

`sprintf` function ➤22.8

```
char str[20];
int i;
...
sprintf(str, "%d", i);    /* writes i into the string str */
```

Not only is `sprintf` portable, but it also provides a great deal of control over the appearance of the number.

***Q: The description of the `strtod` function says that C99 allows the string argument to contain a hexadecimal floating-point number, infinity, or NaN. What is the format of these numbers? [p. 684]**

A: A hexadecimal floating-point number begins with `0x` or `0X`, followed by one or more hexadecimal digits (possibly including a decimal-point character), and then possibly a binary exponent. (See the Q&A at the end of Chapter 7 for a discussion of hexadecimal floating constants, which have a similar—but not identical—format.) Infinity has the form `INF` or `INFINITY`; any or all of the letters may be lower-case. NaN is represented by the string `NAN` (again ignoring case), possibly followed by a pair of parentheses. The parentheses may be empty or they may contain a series of characters, where each character is a letter, digit, or underscore. The characters may be used to specify some of the bits in the binary representation of the NaN value, but their exact meaning is implementation-defined. The same kind of character sequence—which the C99 standard calls an *n-char-sequence*—is also used in calls of the `nan` function.

`nan` function ➤23.4

***Q: You said that performing the call `exit(`*n*`)` anywhere in a program is *normally* equivalent to executing the statement `return` *n*`;` in `main`. When would it not be equivalent? [p. 688]**

A: There are two issues. First, when the `main` function returns, the lifetime of its local variables ends (assuming that they have automatic storage duration, as they will unless they're declared to be `static`), which isn't true if the `exit` function is called. A problem will occur if any action that takes place at program termination—such as calling a function previously registered using `atexit` or flushing an output stream buffer—requires access to one of these variables. In particular, a program might have called `setvbuf` and used one of `main`'s variables as a buffer. Thus, in rare cases a program may behave improperly if it attempts to return from `main` but work if it calls `exit` instead.

automatic storage duration ➤18.2

`setvbuf` function ➤22.2

C99

The other issue occurs only in C99, which makes it legal for `main` to have a return type other than `int` if an implementation explicitly allows the programmer to do so. In these circumstances, the call `exit(`*n*`)` isn't necessarily equivalent to executing `return` *n*`;` in `main`. In fact, the statement `return` *n*`;` may be illegal (if `main` is declared to return `void`, for example).