The statement

```
return 0;
```

has two effects: it causes the `main` function to terminate (thus ending the program) and it indicates that the `main` function returns a value of 0. We'll have more to say about `main`'s return value in a later chapter. For now, we'll always have `main` return the value 0, which indicates normal program termination.

return value of `main` ➤ 9.5

Q&A

If there's no `return` statement at the end of the `main` function, the program will still terminate. However, many compilers will produce a warning message (because the function was supposed to return an integer but failed to).

Q&A

Statements

A *statement* is a command to be executed when the program runs. We'll explore statements later in the book, primarily in Chapters 5 and 6. The `pun.c` program uses only two kinds of statements. One is the `return` statement; the other is the *function call*. Asking a function to perform its assigned task is known as *calling* the function. The `pun.c` program, for example, calls the `printf` function to display a string on the screen:

```
printf("To C, or not to C: that is the question.\n");
```

C requires that each statement end with a semicolon. (As with any good rule, there's one exception: the compound statement, which we'll encounter later.) The semicolon shows the compiler where the statement ends; since statements can continue over several lines, it's not always obvious where they end. Directives, on the other hand, are normally one line long, and they *don't* end with a semicolon.

compound statement ➤ 5.2

Printing Strings

`printf` is a powerful function that we'll examine in Chapter 3. So far, we've only used `printf` to display a ***string literal***—a series of characters enclosed in double quotation marks. When `printf` displays a string literal, it doesn't show the quotation marks.

`printf` doesn't automatically advance to the next output line when it finishes printing. To instruct `printf` to advance one line, we must include `\n` (the *new-line character*) in the string to be printed. Writing a new-line character terminates the current output line; subsequent output goes onto the next line. To illustrate this point, consider the effect of replacing the statement

```
printf("To C, or not to C: that is the question.\n");
```

by two calls of `printf`:

```
printf("To C, or not to C: ");
printf("that is the question.\n");
```