Functions

Functions are like "procedures" or "subroutines" in other programming languages—they're the building blocks from which programs are constructed. In fact, a C program is little more than a collection of functions. Functions fall into two categories: those written by the programmer and those provided as part of the C implementation. I'll refer to the latter as *library functions*, since they belong to a "library" of functions that are supplied with the compiler.

The term "function" comes from mathematics, where a function is a rule for computing a value when given one or more arguments:

$$f(x) = x + 1$$
$$g(y, z) = y^2 - z^2$$

C uses the term "function" more loosely. In C, a function is simply a series of statements that have been grouped together and given a name. Some functions compute a value; some don't. A function that computes a value uses the `return` statement to specify what value it "returns." For example, a function that adds 1 to its argument might execute the statement

```
return x + 1;
```

while a function that computes the difference of the squares of its arguments might execute the statement

```
return y * y - z * z;
```

Although a C program may consist of many functions, only the `main` function is mandatory. `main` is special: it gets called automatically when the program is executed. Until Chapter 9, where we'll learn how to write other functions, `main` will be the only function in our programs.

The name `main` is critical; it can't be `begin` or `start` or even `MAIN`.

If `main` is a function, does it return a value? Yes: it returns a status code that is given to the operating system when the program terminates. Let's take another look at the `pun.c` program:

```
#include <stdio.h>

int main(void)
{
  printf("To C, or not to C: that is the question.\n");
  return 0;
}
```

The word `int` just before `main` indicates that the `main` function returns an integer value. The word `void` in parentheses indicates that `main` has no arguments.