C Basics: Statements and Functions

Every programming language is designed to follow strict rules that define the language’s source code structure. The C programming language is no different. These next few chapters will explore the syntax of C. This chapter focuses on two of the primary building blocks of C programming: statements and functions. In a nutshell, a statement tells the computer to do something. A function is a series of statements.

C Statements

A statement in C is very much like a declarative statement in English; it tells the computer to do something. “Say ‘Hello’” and “preheat oven to 350°F” are examples of concise, unambiguous, English statements. Here are two statements in the C language:

```c
printf( "Hello!\n" );
temperature = 350;
```

The first statement tells the computer to make the text “Hello!” appear on the console (similar to what you saw in Chapter 2). The second statement tells the computer to assign the value 350 to a variable named `temperature`. C statements end with a semicolon (;), just as English sentences end with a period.
C Functions

A C function is a group of C statements. There are many reasons for organizing statements into a function, but the primary reason for gathering statements into a single function is to make them easily reusable.

A cake recipe consists of many individual steps: “sift flour,” “add eggs,” “preheat oven,” “spread icing,” and so on. By organizing these steps into a larger entity, a recipe, we can now simply say “bake a cake,” instead of repeating all of the individual steps.

Similarly, a C function is a sequence of C statements, the whole of which can be invoked as a C statement. (Read that again, slowly.) You create your own functions in C like this:

```c
void SayHello( void )
{
    printf( "Hello!!!\n" );
}
```

You just created a function named SayHello(), which does one thing. It consists of a single statement that calls another function, named printf(), that outputs a message to the console window.

CONSOLE

Technically, the function printf() sends its output to something called standard output and Xcode redirects standard output to its console pane. You’ll learn more about standard output in Chapter 9 when we discuss the command line. For the moment, just think of printf() as a function that sends information to the console.

NOTE: Throughout this book, we’ll designate a function by placing a pair of parentheses after its name. This will help distinguish between variable names and function names. For example, the name doTask refers to a variable (variables are covered in Chapter 4), while doTask() refers to a function.

The printf() function consists of dozens of statements, many of which call other functions, which themselves consist of dozens of statements, many of which call even more functions, and so on—unraveling computer programs can be a lot like peeling an onion. The point is, you don’t need to concern yourself with