1.1 — Statements and the structure of a program

Chapter introduction

Welcome to the first primary chapter of these C++ tutorials!

In this chapter, we'll take a first look at a number of topics that are essential to every C++ program. Because there are quite a few topics to cover, we'll cover most at a fairly shallow level (just enough to get by). The goal of this chapter is to help you understand how basic C++ programs are constructed. By the end of the chapter, you will be able to write your own simple programs.

In future chapters, we'll revisit the majority of these topics and explore them in more detail. We'll also introduce new concepts that build on top of these.

In order to keep the lesson lengths manageable, topics may be split over several subsequent lessons. If you feel like some important concept isn't covered in a lesson, it's possible that it's covered in the next lesson.

Statements

A computer program is a sequence of instructions that tell the computer what to do. A **statement** is a type of instruction that causes the program to *perform some action*.

Statements are by far the most common type of instruction in a C++ program. This is because they are the smallest independent unit of computation in the C++ language. In that regard, they act much like sentences do in natural language. When we want to convey an idea to another person, we typically write or speak in sentences (not in random words or syllables). In C++, when we want to have our program do something, we typically write statements.

Most (but not all) statements in C++ end in a semicolon. If you see a line that ends in a semicolon, it's probably a statement.

In a high-level language such as C++, a single statement may compile into many machine language instructions.

For advanced readers

There are many different kinds of statements in C++:

- Declaration statements

- Jump statements
- Expression statements
- Compound statements
- Selection statements (conditionals)
- Iteration statements (loops)
- Try blocks

By the time you're through with this tutorial series, you'll understand what all of these are!

Functions and the main function

In C++, statements are typically grouped into units called functions. A **function** is a collection of statements that get executed sequentially (in order, from top to bottom). As you learn to write your own programs, you'll be able to create your own functions and mix and match statements in any way you please (we'll show how in a future lesson).

Rule

Every C++ program must have a special function named **main** (all lower case letters). When the program is run, the statements inside of `main` are executed in sequential order.

Programs typically terminate (finish running) after the last statement inside function `main` has been executed (though programs may abort early in some circumstances, or do some cleanup afterwards).

Functions are typically written to do a specific job or perform some useful action. For example, a function named `max` might contain statements that figures out which of two numbers is larger. A function named `calculateGrade` might calculate a student's grade from a set of test scores. A function named `printEmployee` might print an employee's information to the console. We will talk a lot more about functions soon, as they are the most commonly used organizing tool in a program.

Nomenclature

When discussing functions, it's fairly common shorthand to append a pair of parenthesis to the end of the function's name. For example, if you see the term `main()` or `doSomething()`, this is shorthand for functions named `main` or `doSomething` respectively. This helps differentiate functions from other things with names (such as variables) without having to write the word "function" each time.

In programming, the name of a function (or object, type, template, etc…) is called its **identifier**.

Dissecting Hello world!

Now that you have a brief understanding of what statements and functions are, let's return to our "Hello world" program and take a high-level look at what each line does in more detail.

```
#include <iostream>

int main()
{
   std::cout << "Hello world!";
   return 0;
}
```

Line 1 is a special type of line called a preprocessor directive. This preprocessor directive indicates that we would like to use the contents of the `iostream` library, which is the part of the C++ standard library that allows us to read and write text from/to the console. We need this line in order to use `std::cout` on line 5. Excluding this line would result in a compile error on line 5, as the compiler wouldn't otherwise know what `std::cout` is.

Line 2 is blank, and is ignored by the compiler. This line exists only to help make the program more readable to humans (by separating the `#include` preprocessor directive and the subsequent parts of the program).

Line 3 tells the compiler that we're going to write (define) a function whose name (identifier) is `main`. As you learned above, every C++ program must have a `main` function or it will fail to link.

Lines 4 and 7 tell the compiler which lines are part of the *main* function. Everything between the opening curly brace on line 4 and the closing curly brace on line 7 is considered part of the `main` function. This is called the function body.

Line 5 is the first statement within function `main`, and is the first statement that will execute when we run our program. `std::cout` (which stands for "character output") and the `<<` operator allow us to display information on the console. In this case, we're displaying the text "Hello world!". This statement creates the visible output of the program.

Line 6 is a return statement. When an executable program finishes running, the program sends a value back to the operating system in order to indicate whether it ran successfully or not. This particular return statement returns the value `0` to the operating system, which means "everything went okay!". This is the last statement in the program that executes.

All of the programs we write will follow this general template, or a variation on it.

Author's note

If parts (or all) of the above explanation are confusing, that's to be expected at this point. This was just to provide a quick overview. Subsequent lessons will dig into all of the above topics, with plenty of additional explanation and examples.

You can compile and run this program yourself, and you will see that it outputs the following to the console:

```
Hello world!
```

If you run into issues compiling or executing this program, check out lesson 0.8 -- A few common C++ problems.

Syntax and syntax errors

In English, sentences are constructed according to specific grammatical rules that you probably learned in English class in school. For example, normal sentences end in a period. The rules that govern how sentences are constructed in a language is called **syntax**. If you forget the period and run two sentences together, this is a violation of the English language syntax.

C++ has a syntax too: rules about how your programs must be constructed in order to be considered valid. When you compile your program, the compiler is responsible for making sure your program follows the basic syntax of the C++ language. If you violate a rule, the compiler will complain when you try to compile your program, and issue you a **syntax error**.

Let's see what happens if we omit the semicolon on line 5 of the "Hello world" program, like this:

```
#include <iostream>

int main()
{
   std::cout << "Hello world!"
   return 0;
}
```

Feel free to compile this ill-formed program yourself.

Visual Studio produces the following error (your compiler may generate an error message with different wording):

```
c:\vcprojects\hello.cpp(6): error C2143: syntax error : missing ';' before 'return'
```

The compiler is telling you that it encountered a syntax error on line 6: it was expecting a semicolon before the return statement, but it didn't find one. Although the compiler will tell you which line of code it was compiling when it encountered the syntax error, the thing that needs to be fixed may actually be on a previous line. In this case, we'd conventionally place the semicolon at the end of line 5.

Syntax errors are common when writing a program. Fortunately, they're typically straightforward to find and fix, as the compiler will generally point you right at them. Compilation of a program will only complete once all syntax errors are resolved.

You can try deleting characters or even whole lines from the "Hello world" program to see different kinds of errors that get generated. Try restoring the missing semicolon at the end of line 5, and then deleting lines 1, 3, or 4 and see what happens.

Quiz time

The following quiz is meant to reinforce your understanding of the material presented above.

Question #1

What is a statement?

Show Solution

Question #2

What is a function?

Show Solution

Question #3

What is the name of the function that all programs must have?

Show Solution

Question #4

When a program is run, where does execution start?

Show Solution

Question #5

What symbol are statements in C++ often ended with?

Show Solution

Question #6

What is a syntax error?

Show Solution

Question #7

What is the C++ Standard Library?

Show Hint

Show Solution