1.1 — Structure of a program

BY ALEX ON MAY 30TH, 2007 | LAST MODIFIED BY ALEX ON APRIL 10TH, 2017

A computer program is a sequence of instructions that tell the computer what to do.

Statements

The most common type of instruction in a program is the **statement**. A statement in C++ is the smallest independent unit in the language. In human language, it is analogous to a sentence. We write sentences in order to convey an idea. In C++, we write statements in order to convey to the compiler that we want to perform a task. **Statements in C++ are often terminated by a semicolon**.

Comment [Office1]: Câu lệnh phải kết thúc bởi dấu ;

There are many different kinds of statements in C++. The following are some of the most common types of simple statements:

```
1 int x;
2 x = 5;
3 std::cout << x;
```

`int x;` is a **declaration statement**. This particular declaration statement tells the compiler that x is a variable that holds an integer (int) value. In programming, a variable provides a name for a region of memory that can hold a value that can vary. All variables in a program must be declared before they are used. We will talk more about variables shortly.

`x = 5;` is an **assignment statement**. It assigns a value (5) to a variable (x).

`std::cout << x;` is an **output statement**. It outputs the value of x (which we set to 5 in the previous statement) to the screen.

Expressions

The compiler is also capable of resolving expressions. An **expression** is a mathematical entity that evaluates to a value. For example, in math, the expression 2+3 evaluates to the value 5. Expressions can involve values (such as 2), variables (such as x), operators (such as +) and functions (which return an output value based on some input value). They can be singular (such as 2, or x), or compound (such as 2+3, 2+x, x+y, or (2+x)*(y-3)).

For example, the statement `x = 2 + 3;` is a valid assignment statement. The expression 2 + 3 evaluates to the value of 5. This value of 5 is then assigned to x.

Functions

Comment [Office2]: Các câu lệnh thường được nhóm thành hàm. Bất cứ chương trình nào đều có hàm đặc biệt là hàm main.

In C++, statements are typically grouped into units called functions. A **function** is a collection of statements that executes sequentially. Every C++ program must contain a special function called **main**. When the C++ program is run, execution starts with the first statement inside of function main. Functions are typically written to do a very specific job. 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. We will talk more about functions later.

Helpful hint: It's a good idea to put your main() function in a .cpp file named either main.cpp, or with the same name as your project. For example, if you are writing a Chess game, you could put your main() function in chess.cpp.

Libraries and the C++ Standard Library

A **library** is a collection of precompiled code (e.g. functions) that has been "packaged up" for reuse in many different programs. Libraries provide a common way to extend what your programs can do. For example, if you were writing a game, you'd probably want to include a sound library and a graphics library.

The C++ core language is actually very small and minimalistic (and you'll learn most of it in these tutorials). However, C++ also comes with a library called the **C++ standard library** that provides additional functionality for your use. The C++ standard library is divided into areas (sometimes also called libraries, even though they're just parts of the standard library), each of which focus on providing a specific type of functionality. One of the most commonly used parts of the C++ standard library is the iostream library, which contains functionality for writing to the screen and getting input from a console user.

Taking a look at a sample program

Now that you have a brief understanding of what statements, functions, and libraries are, let's look at a simple "hello world" program:

```
1 #include <iostream>
2
3 int main()
4 {
5    std::cout << "Hello world!";
6    return 0;
7 }
```

Line 1 is a special type of statement called a **preprocessor directive**. Preprocessor directives tell the compiler to perform a special task. In this case, we are telling the compiler that we would like to add the contents of the iostream header to our program.

The iostream header allows us to access functionality from the iostream library, which will allow us to write text to the screen.

Line 2 is blank, and is ignored by the compiler.

Line 3 declares the main() function, which as you learned above, is mandatory. Every program must have a main() function.

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.

Line 5 is our first statement (you can tell it's a statement because it ends with a semicolon), and it is an output statement. std::cout is a special object that represents the console/screen. The << symbol is an operator (much like + is an operator in mathematics) called the **output operator**. std::cout understands that anything sent to it via the output operator should be printed on the screen. In this case, we're sending it the text "Hello world!".

Line 6 is a new type of statement, called a **return statement**. When an executable program finishes running, the main() function sends a value back to the operating system that indicates whether it was run successfully or not.

This particular return statement returns the value of 0 to the operating system, which means "everything went okay!". Non-zero numbers are typically used to indicate that something went wrong, and the program had to abort. We will discuss return statements in more detail when we discuss functions.

All of the programs we write will follow this template, or a variation on it. We will discuss each of the lines above in more detail in the upcoming sections.

(Remember, Visual Studio users should add #include "stdafx.h" as the first line of any C++ code file written in Visual Studio)

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**.

Comment [Office3]: Các câu lệnh trong chương trình phải đúng cú pháp.

For example, you learned above that statements must end in a semicolon.

Let's see what happens if we omit the semicolon in the following program:

```
1 #include <iostream>
2
3 int main()
4 {
5    std::cout << "Hello world!"
6    return 0;
7 }
```

Visual studio produces the following error:

```
c:\users\apomeranz\documents\visual studio 2013\projects\test1\test1\test1.cpp(6): error C2143: syntax error : missing ';' before 'return'
```

This is telling you that you have an syntax error on line 6: You've forgotten a semicolon before the return. In this case, the error is actually at the end of line 5. Often, the compiler will pinpoint the exact line where the syntax error occurs for you. However, sometimes it doesn't notice until the next line.

Syntax errors are common when writing a program. Fortunately, they're often easily fixable. The program can only be fully compiled (and executed) once all syntax errors are resolved.