Structure of a program

The best way to learn a programming language is by writing programs. Typically, the first program beginners write is a program called "Hello World", which simply prints "Hello World" to your computer screen. Although it is very simple, it contains all the fundamental components C++ programs have:

学习编程语言的最好方法就是编写程序。通常，初学者编写的第一个程序是一个叫做“Hello World”的程序，它简单地将“Hello World”打印到你的计算机屏幕上。虽然它非常简单，但它包含了c++程序所拥有的所有基本组件:

```
// my first program in C++
#include <iostream>

int main()
{
  std::cout << "Hello World!";
}
```

The left panel above shows the C++ code for this program. The right panel shows the result when the program is executed by a computer. The grey numbers to the left of the panels are line numbers to make discussing programs and researching errors easier. They are not part of the program.

上面的左面板显示了这个程序的c++代码。右边的面板显示了当程序被计算机执行时的结果。面板左边的灰色数字是行号，使讨论程序和研究错误更容易。他们不是这个项目的一部分。

Let's examine this program line by line:

让我们逐行检查这个程序:

- Line 1: `// my first program in C++`

  Two slash signs indicate that the rest of the line is a comment inserted by the programmer but which has no effect on the behavior of the program. Programmers use them to include short explanations or observations concerning the code or program. In this case, it is a brief introductory description of the program.

  两个斜杠表示这一行的其余部分是程序员插入的注释，但对程序的行为没有影响。程序员使用它们来包含关于代码或程序的简短解释或观察。在本例中，它是对该程序的简要介绍。

- Line 2: `#include <iostream>`

  Lines beginning with a hash sign (`#`) are directives read and interpreted by what is known as the *preprocessor*. They are special lines interpreted before the compilation of the program itself begins. In this case, the directive `#include <iostream>`, instructs the preprocessor to include a section of standard C++ code, known as *header iostream*, that allows to perform standard input and output operations, such as writing the output of this program (Hello World) to the screen.

  以#开头的行是由所谓的预处理器读取和解释的指令。它们是在程序本身开始编译之前解释的特殊行。在这种情况下，指令#include 指示预处理器包含一段标准的c++代码，称为头iostream，允许执行标准的输入和输出操作，比如将这个程序(Hello World)的输出写到屏幕上。

- Line 3: A blank line.

  Blank lines have no effect on a program. They simply improve readability of the code.

  空行对程序没有影响。它们只是提高了代码的可读性。

- Line 4: `int main ()`

  This line initiates the declaration of a function. Essentially, a function is a group of code statements which are given a name: in this case, this gives the name "main" to the group of code statements that follow. Functions will be discussed in detail in a later chapter, but essentially, their definition is introduced with a succession of a type (`int`), a name (`main`) and a pair of parentheses (`()`), optionally including parameters.

  这一行开始了函数的声明。本质上，函数是一组给定名称的代码语句:在本例中，为后面的一组代码语句赋予了名称“main”。函数将在后面的章节中详细讨论，但本质上，它们的定义是通过一个类型(int)、一个名称(main)和一对圆括号(())(可选地包括参数)来介绍的。

  The function named `main` is a special function in all C++ programs; it is the function called when the program is run. The execution of all C++ programs begins with the `main` function, regardless of where the function is actually located within the code.

  main函数是所有c++程序中的一个特殊函数;它是程序运行时调用的函数。所有c++程序的执行都从main函数开始，而不管main函数在代码中的实际位置。

- Lines 5 and 7: `{` and `}`

  The open brace (`{`) at line 5 indicates the beginning of `main`'s function definition, and the closing brace (`}`) at line 7, indicates its end. Everything between these braces is the function's body that defines what happens when `main` is called. All functions use braces to indicate the beginning and end of their definitions.

  第5行左括号('{')表示' main '函数定义的开始，第7行右括号('}')表示其结束。这些大括号之间的所有内容都是函数体，它定义了调用' main '时会发生什么。所有函数都使用大括号来表示它们定义的开始和结束。

- Line 6: `std::cout << "Hello World!";`

This line is a C++ statement. A statement is an expression that can actually produce some effect. It is the meat of a program, specifying its actual behavior. Statements are executed in the same order that they appear within a function's body.

这一行是一个c++语句。语句是一种可以实际产生某种效果的表达式。它是程序的核心，指定了它的实际行为。语句的执行顺序与它们在函数体中出现的顺序相同。

This statement has three parts: First, `std::cout`, which identifies the **st**andar**d** **c**haracter **out**put device (usually, this is the computer screen). Second, the insertion operator (`<<`), which indicates that what follows is inserted into `std::cout`. Finally, a sentence within quotes ("Hello world!"), is the content inserted into the standard output.

该语句有三部分:第一部分是std::cout，它标识标准字符输出设备(通常是计算机屏幕)。第二，插入操作符(<<)，它表示下面的内容被插入到std::cout中。最后，引号内的句子(“Hello world!”)是插入到标准输出中的内容。

Notice that the statement ends with a semicolon (`;`). This character marks the end of the statement, just as the period ends a sentence in English. All C++ statements must end with a semicolon character. One of the most common syntax errors in C++ is forgetting to end a statement with a semicolon.

注意，该语句以分号(;)结尾。这个字符标志着语句的结束，就像英语中的句号一样。所有c++语句都必须以分号字符结束。c++中最常见的语法错误之一是忘记用分号结束语句。

You may have noticed that not all the lines of this program perform actions when the code is executed. There is a line containing a comment (beginning with `//`). There is a line with a directive for the preprocessor (beginning with `#`). There is a line that defines a function (in this case, the `main` function). And, finally, a line with a statements ending with a semicolon (the insertion into `cout`), which was within the block delimited by the braces (`{ }`) of the `main` function.

您可能已经注意到，在执行代码时，并不是该程序的所有行都执行操作。有一行包含注释(以//开头)。有一行是给预处理器的指令(以#开头)。其中一行定义了一个函数(在本例中是main函数)。最后一行，语句以分号结尾(插入cout)，它位于主函数的大括号({})分隔的块中。

The program has been structured in different lines and properly indented, in order to make it easier to understand for the humans reading it. But C++ does not have strict rules on indentation or on how to split instructions in different lines. For example, instead of

该程序被构造成不同的行，并正确地缩进，以便阅读它的人更容易理解。但是c++对于缩进或如何在不同的行中分割指令没有严格的规则。例如，代替

```
int main ()
{
  std::cout << " Hello World!";
}
```

We could have written:

```
int main () { std::cout << "Hello World!"; }
```

all in a single line, and this would have had exactly the same meaning as the preceding code.

所有这些都在一行中，这将具有完全相同的意义，正如前面的代码。

In C++, the separation between statements is specified with an ending semicolon (`;`), with the separation into different lines not mattering at all for this purpose. Many statements can be written in a single line, or each statement can be in its own line. The division of code in different lines serves only to make it more legible and schematic for the humans that may read it, but has no effect on the actual behavior of the program.

在c++中，语句之间的分隔用一个结束分号(';')来指定，对于这个目的，不同行之间的分隔根本无关紧要。许多语句可以写在一行中，或者每个语句可以写在自己的行中。将代码划分为不同的行只是为了让人们更容易理解和理解它，但对程序的实际行为没有影响。

Now, let's add an additional statement to our first program:

现在，让我们向第一个程序添加一个附加语句:

```
// my second program in C++
#include <iostream>

int main ()
{
  std::cout << "Hello World! ";
  std::cout << "I'm a C++ program";
}
```

In this case, the program performed two insertions into `std::cout` in two different statements. Once again, the separation in different lines of code simply gives greater readability to the program, since `main` could have been perfectly valid defined in this way:

在本例中，程序在两个不同的语句中对' std::cout '执行了两次插入。再一次，不同代码行之间的分离给程序带来了更大的可读性，因为“main”可以这样定义:

```
int main () { std::cout << " Hello World! "; std::cout << " I'm a C++ program ";
```

The source code could have also been divided into more code lines instead:

源代码也可以被分成更多的代码行:

```
int main ()
{
  std::cout <<
    "Hello World!";
  std::cout
    << "I'm a C++ program";
}
```

And the result would again have been exactly the same as in the previous examples.

结果和前面的例子完全一样。

Preprocessor directives (those that begin by `#`) are out of this general rule since they are not statements. They are lines read and processed by the preprocessor before proper compilation begins. Preprocessor directives must be specified in their own line and, because they are not statements, do not have to end with a semicolon (`;`).

预处理器指令(那些以“#”开头的指令)不属于这一通用规则，因为它们不是语句。它们是在正确的编译开始之前由预处理程序读取和处理的行。预处理器指令必须在它们自己的行中指定，因为它们不是语句，所以不必以分号(';')结束。

Comments

As noted above, comments do not affect the operation of the program; however, they provide an important tool to document directly within the source code what the program does and how it operates.

如上所述，注释并不影响程序的运行;但是，它们提供了一个重要的工具，可以在源代码中直接记录程序的功能和运行方式。

C++ supports two ways of commenting code:

c++支持两种注释代码的方式:

```
// line comment
/* block comment */
```

The first of them, known as *line comment*, discards everything from where the pair of slash signs (`//`) are found up to the end of that same line. The second one, known as *block comment*, discards everything between the `/*` characters and the first appearance of the `*/` characters, with the possibility of including multiple lines.

第一个被称为*行注释*，它丢弃了从斜杠符号(' // ')到同一行末尾的所有内容。第二种称为*块注释*，它丢弃' /* '字符和' */ '字符第一次出现之间的所有内容，可能包含多行。

Let's add comments to our second program:

让我们向第二个程序添加注释:

```
/* my second program in C++
   with more comments */

#include <iostream>

int main ()
{
  std::cout << "Hello World! ";     // prints Hello World!
  std::cout << "I'm a C++ program"; // prints I'm a C++ program
}
```

If comments are included within the source code of a program without using the comment characters combinations `//`, `/*` or `*/`, the compiler takes them as if they were C++ expressions, most likely causing the compilation to fail with one, or several, error messages.

如果注释包含在程序的源代码中，而没有使用注释字符组合' // '、' /* '或' */ '，编译器将它们视为c++表达式，很可能导致编译失败，产生一个或多个错误消息。

Using namespace std

If you have seen C++ code before, you may have seen `cout` being used instead of `std::cout`. Both name the same object: the first one uses its *unqualified name* (`cout`), while the second qualifies it directly within the *namespace* `std` (as `std::cout`).

如果你以前看过c++代码，你可能见过' cout '被用来代替' std::cout '。两者命名相同的对象:第一个使用它的*非限定名称*(' cout ')，而第二个直接在*命名空间*' std '中限定它(as ' std::cout ')。

`cout` is part of the standard library, and all the elements in the standard C++ library are declared within what is called a *namespace*: the namespace `std`.

' cout '是标准库的一部分，标准c++库中的所有元素都在所谓的*命名空间*中声明:命名空间' std '。

In order to refer to the elements in the `std` namespace a program shall either qualify each and every use of elements of the library (as we have done by prefixing `cout` with `std::`), or introduce visibility of its components. The most typical way to introduce visibility of these components is by means of *using declarations*:

为了引用' std '命名空间中的元素，程序要么限定每次对标准库元素的使用(就像我们在' cout '前面加上' std:: '那样)，要么引入其组件的可视性。引入这些组件可见性的最典型方法是*using申明*:

```
using namespace std;
```

The above declaration allows all elements in the `std` namespace to be accessed in an *unqualified* manner (without the `std::` prefix).

上面的声明允许以*非限定*的方式访问' std '名称空间中的所有元素(没有' std:: '前缀)。

With this in mind, the last example can be rewritten to make unqualified uses of `cout` as:

记住这一点，可以重写最后一个示例，使' cout '的非限定用法:

```
// my second program in C++
#include <iostream>
using namespace std;

int main ()
{
  cout << "Hello World! ";
  cout << "I'm a C++ program";
}
```

Both ways of accessing the elements of the `std` namespace (explicit qualification and *using* declarations) are valid in C++ and produce the exact same behavior. For simplicity, and to improve readability, the examples in these tutorials will more often use this latter approach with *using* declarations, although note that *explicit qualification* is the only way to guarantee that name collisions never happen.

访问' std '命名空间元素的两种方式(显式限定和*using*声明)在c++中都是有效的，并产生完全相同的行为。为了简单和提高可读性，本教程中的示例将更多地使用后一种方法和*using*声明，不过请注意，*显式限定*是保证名称冲突永远不会发生的唯一方法。

Namespaces are explained in more detail in a later chapter.

名称空间将在后面的章节中进行更详细的解释。