1.1 — Statements and the structure of a program

BY ALEX ON MAY 30TH, 2007 | LAST MODIFIED BY NASCARDRIVER ON DECEMBER 31ST, 2019

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++:

1. Declaration statements
2. Jump statements
3. Expression statements
4. Compound statements
5. Selection statements (conditionals)
6. Iteration statements (loops)
7. 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 executes sequentially. 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, execution starts with the first statement inside of function *main* and then continues sequentially.

Programs typically terminate (finish running) when the last statement inside function *main* is executed (though they may abort early in some circumstances).

Functions are typically written to do a 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 from a set of test scores. We will talk a lot more about functions soon, as they are the most commonly used organizing tool in a program.

Author's note

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 kinds of objects (such as variables) without having to write the word "function" each time.

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.

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

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 called *main*. As you learned above, every C++ program must have a *main* function or it will fail to compile.

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 send letters or numbers to

the console to be output. In this case, we're sending it the text "Hello world!", which will be output to the console. 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 of 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:

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

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\test1.cpp(6): error C2143: syntax error : missing ';' before 'return'
```

This is telling you that you have a syntax error on line 6: the compiler 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 omission may actually be on a previous line. In this case, the error is actually at the end of line 5 (the compiler didn't discover the issue until line 6).

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

1.2 -- Comments

Index

0.12 -- Configuring your compiler: Choosing a language standard

C++ TUTORIAL | PRINT THIS POST

325 comments to 1.1 — Statements and the structure of a program

« Older Comments 1 ... 4 5 6

krishna
January 13, 2020 at 7:14 am · Reply

why is int used to declare main section ....cant we use void instead as it when it used as a declaration statement it is more universal??

nascardriver
January 13, 2020 at 7:20 am · Reply

`main`'s return value is used by other programs to determine if execution of your program was successful (`return 0` means success, everything else means failure)

Vitaliy Sh.
December 31, 2019 at 5:14 am · Reply

List of possible typos:

"We'll also introduce new concepts that build on top of these."
** 'built' && 'on the top' ?

"Excluding this line would result in a compile error ..."
** 'compiler' (text in Question #6 solution: '... is a compiler error ...') ?

"std::cout (which stands for “character output”) and the operator << ..."
** 'operator<<' (with an in-place comment that it's OK)?

Question #3 - self:
"What is the name of the function that all program must have?"
** 'programs' || 'every program'?

Question #5 - solution:
"The semicolon (;)" -- small, hard to visually distinguish from a colon.
**

```
<b style="font-size: 3em">;</b>
```

(please, that is not called 'semicolon' in every language...) ?

Question #7 - solution:
"The C++ Standard Library is a library that ships with C++ that contains additional functionality to use in your programs."
** 'The C++ Standard Library ships with C++. That library contains ...'
('... that ships with C++ that ...' is confusing)?

```
➥
Wish to your all to have a happy celebrations, sires and comrades!
```

nascardriver
December 31, 2019 at 6:12 am · Reply

> build/built
The sentence is correct as is.

> compile/compiler error
Both are fine. There are too many lessons to try to achieve perfect consistency.

> operator< < No quotes needed, but I changed it to "operator <<". That way around it makes more sense until operator overloads are introduced. > all program
Fixed.

> The semicolon
Added code tags, they make it a little bigger.

> The C++ Standard Library
Split into 2 sentences.

Thanks for your suggestions, happy new year!

Vitaliy Sh.
December 31, 2019 at 1:58 am · Reply

Maybe your, sires, willing to change quizzes Show/Hide buttons?

I'm know nothing about Java, but want to implement show/hide functionality in my p. use copy of that site for the quizzes.

In the upper right corner of https://alpinelinux.org/, there is a drop-down menu, without any Java (that i didn't learned yet).

Based on that above mentioned page, and the different Web sources (mozilla's MDN included), i somehow glued HTML to what is below.

I edited "Question 7" from here with the help of my web-browser to show the difference between original and Java-less implementation. No, off course, Java can more, but for that particular anchors (Show/Hide) thingy...

```
<html><head><meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<style>
  .hint_div, .solution_div {
    border-left: 0.1rem solid #0a7;
    padding-left: 0.8rem;
    display: none;
  }
  input:checked ~ .hint_div {display: block;}
  input:checked ~ .solution_div {display: block;}
</style>


</head><body><div class="quiz" style="clear: both">
  <p class="quiz-header">☠ Question #7</p>
  <p></p>

  <ul>
    <p>👁 What is the C++ Standard Library?</p>
  </ul>

<!--<p><a class="hint_link_show" href="javascript:void(0)" onclick="cppHintToggle(document.

  <span>
    <input type="checkbox" id="q7-h1">
      <label for="q7-h1"><a>Show Hint</a></label><!--</p>-->
<!--<div class="hint_div" id="id707129988" style="display:none">-->
    <div class="hint_div" id="id707129988">Review lesson <a href="https://www.learncpp.com/
    </div>
  </span>

  <p></p>

<!--<p><a class="solution_link_show" href="javascript:void(0)" onclick="wpSolutionToggle(do

  <span>
    <input type="checkbox" id="q7-a">
      <label for="q7-a"><a>Show Solution</a></label><!--</p>-->
<!--<div class="solution_div" id="id1474872537" style="display:none">-->
    <div class="solution_div" id="id1474872537">A library file is a collection of precompil
    </div>
  </span>

  <p></p>

</div>

<hr>
</body></html>
```

nascardriver
December 31, 2019 at 6:01 am · Reply

Java is used for Android/Desktop development, JavaScript (JS) is used for web-development. Although they share a part of their name, they're very different.

We don't write the lessons in HTML, we only use HTML for some highlights, eg. and <code>. The other features (eg. links, quizzes) are written with bbcode and transpiled to HTML by the server.

The nice animation on alpinelinux is achieved via CSS.

The current way of showing/hiding the quizzes works, so there's no reason change it.

patrick
December 20, 2019 at 5:06 am · Reply

#include <iostream>

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

I remove the 'int' but the program still work. why?

nascardriver
December 20, 2019 at 5:13 am · Reply

You didn't disable compiler extensions, that's not C++. See lesson 0.10.

Vitaliy Sh.
November 15, 2019 at 9:50 pm · Reply

Hi!

Can your people, please, embed the "forward reference" to the next chapter "1.2 -- Comments" in there, or earlier?

With respect to quizes, nevertheless:
The comments are are useful to express one's understanding of lesson's material by embedding it into the project/solution, somewhere at the bottom of the code.
And then, comparing that with the lesson's definitions.
With the small programs like hello it is more convenient than open an separate file. And that will be inside of a project/solution.

target 1.1_hello of project www_learncpp_com:

```
1  #include <iostream>
2
3  auto main()->int
4  {
5      std::cout << "Hello world!\n"   ;
6      return 0                       ;
7  }
8
9  // It is an C++ one-line comment.
10
11 /* It is an C multi-line comment.
12  * Comments are ignored by compiler,
```

```
 * and are exist for humans to read. */

/* Semicolon in C++ for statements
 * is like a period for sentences
 * in the real English. */

/* Preprocessor directive on line 1
 * tells the compiler to insert there a
 * header file of the iostream library. */

/* The whitespace on line 2 is
 * being ignored by compiler, but
 * enhances the code's readability. */

/* Declaration statement on line 3
 * was not terminated by semicolon.
 * It tells the compiler to expect the
 * definition of the main() within a
 * lines below, between the braces. */

/* Braces on lines 4 and 7 are
 * borders of the main() body. */

/* Statement on line 5 is first that
 * will be executed at the program's
 * runtime, because it is the first
 * statement of the venerable main(). */

/* Statement on line 6 with "return"
 * keyword is the last to be executed
 * within main(). If anything is placed
 * below the "return", it will not be
 * executed. That is useful for the
 * selection statements. */
```

Vitaliy Sh.
November 15, 2019 at 4:59 am · Reply

...
If you feel like some important concept isn't covered in a lesson, it's possible it's covered in the next lesson.
...

Maybe a typo: no "that" after "possible".

Maybe change to:
...
, maybe it's covered in the next lesson.

Todd Riemenschneider
October 2, 2019 at 5:49 pm · Reply

For what its worth anyone using raspian. Here's what I did to run the program.

To compile a file with c++ on raspian

g++ -std=c++0x hello.cpp -ohello

The name of the program to create is specified with the –o flag, in this case we called it hello.
The –std=c++0x flag tells the compiler to use the latest version of the c++ standard.

nascardriver
October 3, 2019 at 12:51 am · Reply

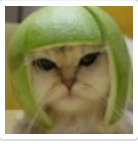
C++0x is the old name for C++11, which isn't the latest version of the standard. You should use the first one from this list that's supported by your version of g++

c++2a
c++17
c++14
c++11

Unit
September 24, 2019 at 12:25 pm · Reply

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

I thought C++ was considered a low-level language, could it be that it's actually a high-level, but is also considered a low-level language?



Alex
September 26, 2019 at 1:31 pm · Reply

C++ is still generally considered a high-level language, but it's definitely at the lower end of that grouping. The terms "high-level" and "low-level" aren't static, and they have been evolving as newer and higher-level languages have been released.

Rusty
September 7, 2019 at 3:43 pm · Reply

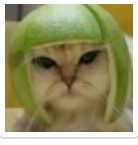
Why is the file location listed out inside the console when I execute the program?

nascardriver
September 8, 2019 at 12:48 am · Reply

Your IDE does that. When you run the program manually, that won't be there.

Mike
September 5, 2019 at 3:49 pm · Reply

Where's the line numbers in Visual Studio 2019? Are they turned off by default now or something.



Alex
September 6, 2019 at 4:26 pm · Reply

They should be on by default. But if they're not, you can turn them on by following the instructions here: https://docs.microsoft.com/en-us/visualstudio/ide/reference/how-to-display-line-numbers-in-the-editor?view=vs-2019

« Older Comments 1 ... 4 5 6