1.11 — Developing your first program

The preceding lessons have introduced a lot of terminology and concepts that we'll use in just about every program we create. In this lesson, we'll walk through the process of integrating this knowledge into our first simple program.

Multiply by 2

First, let's create a program that asks the user to enter an integer, waits for them to input an integer, then tells them what 2 times that number is. The program should produce the following output (assume I entered 4 as input):

```
Enter an integer: 4
Double that number is: 8
```

How do we tackle this? In steps.

Best practice

New programmers often try to write an entire program all at once, and then get overwhelmed when it produces a lot of errors. A better strategy is to add one piece at a time, make sure it compiles, and test it. Then when you're sure it's working, move on to the next piece.

We'll leverage that strategy here. As we go through each step, type (don't copy/paste) each program into your code editor, compile, and run it.

First, create a new console project.

Now let's start with some basic scaffolding. We know we're going to need a main() function (since all C++ programs must have one), so if your IDE didn't create a blank one when you created a new project, let's create one:

```
int main()
{
	return 0;
}
```

We know we're going to need to output text to the console, and get text from the user's keyboard, so we need to include iostream for access to std::cout and std::cin.

```
#include <iostream>

int main()
{
	return 0;
}
```

Now let's tell the user that we need them to enter an integer:

```
#include <iostream>

int main()
{
	std::cout << "Enter an integer: ";

	return 0;
}
```

At this point, your program should produce this result:

```
Enter an integer:
```

and then terminate.

Next, we're going to get the user's input. We'll use std::cin and `operator>>` to get the user's input. But we also need to define a variable to store that input for use later.

```
#include <iostream>

int main() // note: this program has an error somewhere
{
	std::cout << "Enter an integer: ";

	int num{ }; // define variable num as an integer variable
	std::cin << num; // get integer value from user's keyboard

	return 0;
}
```

Time to compile our changes… and…

Uh oh! Here's what the author got on Visual Studio 2017:

```
1>------ Build started: Project: Double, Configuration: Release Win32 ------
1>Double.cpp
1>c:\vcprojects\double\double.cpp(8): error C2678: binary '<<': no operator found
which takes a left-hand operand of type 'std::istream' (or there is no acceptable
conversion)
1>c:\vcprojects\double\double.cpp: note: could be 'built-in C++ operator<<(bool,
int)'
1>c:\vcprojects\double\double.cpp: note: while trying to match the argument list
'(std::istream, int)'
1>Done building project "Double.vcxproj" -- FAILED.
========== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped ==========
```

We ran into a compile error!

First, since the program compiled before we made this latest update, and doesn't compile now, the error *must* be in the code we just added (lines 7 and 8). That significantly reduces the amount of code we have to scan to find the error. Line 7 is pretty straightforward (just a variable definition), so the error probably isn't there. That leaves line 8 as the likely culprit.

Second, this error message isn't very easy to read. But let's pick apart some key elements: The compiler is telling us it ran into the error on line 8. That means the actual error is probably on line 8, or possibly the preceding line, which reinforces our previous assessment. Next, the compiler is telling you that it couldn't find a '<<' operator that has a left-hand operand of type std::istream (which is the type of std::cin). Put another way, operator<< doesn't know what to do with std::cin, so the error must be either with our use of std::cin or our use of operator<<.

See the error now? If you don't, take a moment and see if you can find it.

Here's the program that contains the corrected code:

```
#include <iostream>

int main()
{
	std::cout << "Enter an integer: ";

	int num{ };
	std::cin >> num; // std::cin uses operator >>, not operator <<!

	return 0;
}
```

Now the program will compile, and we can test it. The program will wait for you to enter a number, so let's enter 4. The output should look like this:

```
Enter an integer: 4
```

Almost there! Last step is to double the number.

Once we finish this last step, our program will compile and run successfully, producing the desired output.

There are (at least) 3 ways we can go about this. Let's go from worst to best.

The not-good solution

```
#include <iostream>

// worst version
int main()
{
	std::cout << "Enter an integer: ";

	int num{ };
	std::cin >> num;

	num = num * 2; // double num's value, then assign that value back to num

	std::cout << "Double that number is: " << num << '\n';

	return 0;
}
```

In this solution, we use an expression to multiply *num* by 2, and then assign that value back to *num*. From that point forward, *num* will contain our doubled number.

Why this is a bad solution:

- Before the assignment statement, num contains the user's input. After the assignment, it contains a different value. That's confusing.
- We overwrote the user's input by assigning a new value to the input variable, so if we wanted to extend our program to do something else with that input value later (e.g. triple the user's input), it's already been lost.

The mostly-good solution

```
#include <iostream>

// less-bad version
int main()
{
	std::cout << "Enter an integer: ";

	int num{ };
	std::cin >> num;

	int doublenum{ num * 2 }; // define a new variable and initialize it with num
* 2
	std::cout << "Double that number is: " << doublenum << '\n'; // then print
the value of that variable here

	return 0;
}
```

This solution is pretty straightforward to read and understand, and resolves both of the problems encountered in the worst solution.

The primary downside here is that we're defining a new variable (which adds complexity) to store a value we only use once. We can do better.

The preferred solution

```
#include <iostream>

// preferred version
int main()
{
	std::cout << "Enter an integer: ";

	int num{ };
	std::cin >> num;

	std::cout << "Double that number is: " <<  num * 2 << '\n'; // use an
expression to multiply num * 2 at the point where we are going to print it

	return 0;
}
```

This is the preferred solution of the bunch. When std::cout executes, the expression *num * 2* will get evaluated, and the result will be double *num*'s value. That value will get printed. The value in *num* itself will not be altered, so we can use it again later if we wish.

This version is our reference solution.

Author's note

The first and primary goal of programming is to make your program work. A program that doesn't work isn't useful regardless of how well it's written.

However, there's a saying I'm fond of: "You have to write a program once to know how you should have written it the first time." This speaks to the fact that the best solution often isn't obvious, and that our first solutions to problems are usually not as good as they could be.

When we're focused on figuring out how to make our programs work, it doesn't make a lot of sense to invest a lot of time into code we don't even know if we'll keep. So we take shortcuts. We skip things like error handling and comments. We sprinkle debugging code throughout our solution to help us diagnose issues and find errors. We learn as we go -- things we thought might work don't work after all, and we have to backtrack and try another approach.

The end result is that our initial solutions often aren't well structured, robust (error-proof), readable, or concise. So once your program is working, your job really isn't done (unless the program is a one-off/throwaway). The next step is to cleanup your code. This involves things like: removing (or commenting out) temporary/debugging code, adding comments, handling error cases, formatting your code, and ensuring best practices are followed. And even then, your program may not be as simple as it could be -- perhaps there is redundant logic that can be consolidated, or multiple statements that can be combined, or variables that aren't needed, or a thousand other little things that could be simplified. Too often new programmers focus on optimizing for performance when they should be optimizing for maintainability.

Very few of the solutions presented in these tutorials came out great the first time. Rather, they're the result of continual refinement until nothing else could be found to improve. And in many cases, readers still find plenty of other things to suggest as improvements!

All of this is really to say: don't be frustrated if/when your solutions don't come out wonderfully optimized right out of your brain. That's normal. Perfection in programming is an iterative process (one requiring repeated passes).

Author's note

One more thing: You may be thinking, "C++ has so many rules and concepts. How do I remember all of this stuff?".

Short answer: You don't. C++ is one part using what you know, and two parts looking up how to do the rest.

As you read through this site for the first time, focus less on memorizing specifics, and more on understanding what's possible. Then, when you have a need to implement something in a program you're writing, you can come back here (or to a reference site) and refresh yourself on how to do so.

Quiz time

Question #1

Modify the solution to the “best solution” program above so that it outputs like this (assuming user input 4):

```
Enter an integer: 4
Double 4 is: 8
Triple 4 is: 12
```

Show Solution