1.10 — Developing your first program

BY ALEX ON FEBRUARY 1ST, 2019 | LAST MODIFIED BY NASCARDRIVER ON JANUARY 31ST, 2020

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 compiler, 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++ must have one), so if your IDE didn't create a blank one when you created a new project, let's create one:

```cpp
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.

```cpp
#include <iostream>

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

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

```cpp
#include <iostream>

int main()
{
```

```
5        std::cout << "Enter an integer: ";
6
7        return 0;
8    }
```

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.

```
1    #include <iostream>
2
3    int main()
4    {
5        std::cout << "Enter an integer: ";
6
7        int num{ 0 }; // define variable num as an integer variable
8        std::cin << num; // get integer value from user's keyboard
9
10       return 0;
11   }
```

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 t
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::i
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:

```
1    #include <iostream>
2
3    int main()
```

```cpp
{
    std::cout << "Enter an integer: ";

    int num{ 0 };
    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

```cpp
#include <iostream>

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

    int num{ 0 };
    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

```cpp
#include <iostream>

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

    int num{ 0 };
    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 th

    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{ 0 };
    std::cin >> num;

    std::cout << "Double that number is: " <<  num * 2 << '\n'; // use an expression to multip

    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 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

1.x -- Chapter 1 summary and quiz

Index

1.9 -- Introduction to expressions

C++ TUTORIAL | PRINT THIS POST

38 comments to 1.10 — Developing your first program

Ryan

January 30, 2020 at 5:52 pm · Reply

Hi! I found a couple typos, and had a question to ask. In this chapter, it says "multiple" instead of "multiply" in "we use an expression to multiple num by 2". There is also an unnecessary "a" in "This solution is a pretty straightforward to read and understand". A couple chapters back, in 1.8 "Introduction to literals and operators", it says "flip" instead of "flips" in "takes literal operand 5 and flip its sign".

In the following two examples, would there be *any* difference in performance to any theoretical extent? Especially if done to some exaggerated extreme amount of times? Or do they evaluate to the exact same thing in compilation; and the first would be preferred for organizational purposes?

```
#include <iostream>

int main()
{
    std::cout << "Enter an integer: ";
    int a{};
    std::cin >> a;
    std::cout << "Double " << a << " is: " << 2 * a << '\n';
    std::cout << "Triple " << a << " is: " << 3 * a << '\n';
    return 0;
}
```

```
#include <iostream>

int main()
{
    std::cout << "Enter an integer: ";
    int a{};
    std::cin >> a;
    std::cout << "Double " << a << " is: " << 2 * a << '\n' << "Triple " << a << " is: " <<
    return 0;
}
```

nascardriver

January 31, 2020 at 12:55 am · Reply

Thanks for pointing out the spelling mistakes, all fixed!

Version 2 is faster. `std::cout`'s address has to be loaded into a register (A fast variable on the CPU). In version 1, it has to be loaded twice, whereas version 2 loads it just once, otherwise they're the same. You're not going to notice this, especially since we're talking about IO, which is slow in general.

Vitaliy Sh.
January 25, 2020 at 3:12 am · Reply

Hi Sires!

```
//          C++ programs ?
"(since all C++ must have one)"


//  multipl_Y``
"to multiple num by 2,"


//  doubledNum ?
int doublenum


// https://en.cppreference.com/w/
// https://isocpp.org/get-started
// https://github.com/EbookFoundation/free-programming-books/blob/master/free-programming-b
// or such link?
"(or to a reference site)"


//                 from the "best solution" ?
"Modify the solution to the “best solution” program above..."


//  num{ }; ?
int num{ 0 };
```

Petertheminer11
January 2, 2020 at 4:21 pm · Reply

```
#include <iostream>
int main()
{
   std::cout << "Enter an Integer: ";
   int {user_value = 0};
   std::cin >> user_value;
   std::cout << "Double your integer is: " << user_value * 2 << std::endl;
   std::cout << "Triple your integer is: " << user_value * 3 << std::endl;



   return 0;
}
```

I programmed this, but I'm not sure which category this would fall under (the not-good solution, the ok solution, or the preferred solution).

nascardriver
January 3, 2020 at 6:14 am · Reply

The untested solution.

```
int {user_value = 0}; // This is not C++
int user_value{ 0 }; // These 2 are C++
int user_value{};
```

Also, don't use `std::endl` unless you need to. '\n' should be preferred, as noted in lesson 1.5.

You can highlight your code in the comments by using code tags:
[-code]
your code here
[-/code]
Without the -

Apaulture
January 12, 2020 at 5:10 pm · Reply

```
int user_value{0};
```

is preferred.
But since user_value may be replaced anyway,

```
int user_value{};
```

will also work.

Ryan
November 26, 2019 at 1:00 pm · Reply

Would it be a good habit to initialise like int num{}; or int num{ 0 };

nascardriver
November 27, 2019 at 4:07 am · Reply

If you use the initial value (0), explicitly initialize to 0

```
int num{ 0 };

std::cout << (num + 1) << '\n';
```

If you don't use the value, use value-initialization.

```
int num{};

std::cin >> num;
```

They both do the same, but the first version is easier to understand if the value is used.

Jake
October 2, 2019 at 1:47 pm · Reply

(Please forgive my getting ahead by "using namespace"; also multiple definitions on a line; I's jest set in me ways. :-)
I decided to get a a bit mathematical and added a hypotenuse calculation. Surprising result..

```
#include <iostream>
using namespace std;

int main()
{
  int num1{ 0 }, num2{ 0 };
  cout << "Enter two integers: ";
  cin >> num1 >> num2;
  cout << num1 << " + " << num2 << " = " << num1 + num2 << endl;
  cout << num1 << " - " << num2 << " = " << num1 - num2 << endl;
  float hyp{ 0 };
  hyp = sqrt(num1 ^ 2 + num2 ^ 2);
  cout << "Hypotenuse is " << hyp << endl;
}
```

Result:

```
Enter two integers: 15 8
15 + 8 = 23
15 - 8 = 7
Hypotenuse is 2.64575
```

The result should have been 17 or something VERY close to that. What have I missed?

nascardriver
October 3, 2019 at 12:17 am · Reply

\`^\` is not power. It's a logical XOR (Chapter O (That's an o, not a zero)). If you want power, use \`std::pow\` from the \`<cmath>\` header.

```
float hyp{ std::sqrt(std::pow(num1, 2.0f) + std::pow(num2, 2.0f)) };
```

EDIT: "header" means you have to add \`#include <cmath>\`.

Shubham Maske
September 20, 2019 at 9:19 am · Reply

I am having an issue with code Block software

after running the program the Black screen appears

it says Enter an Integer :
when I type any integer 1 or 2 or say 5
and press enter
the black screen disappears and the result is not shown.

can anyone help me fix this?

nascardriver
September 21, 2019 at 2:05 am · Reply

See lesson 0.7 "If your program runs but the window flashes and closes immediately"

Shubham Maske
September 21, 2019 at 7:39 am · Reply

Nascar driver thanks for the help.

By the way, will I need to follow the same process of adding those 3 lines of code every time I am running a program or not?

Is there any permanent solution for the issue?

nascardriver
September 21, 2019 at 7:41 am · Reply

You've got to look through code block's settings to see if there's an option to keep the console open. Alternatively, you could run your program from the command line (shift+right click in the directory of your executable).

4STR4L_3N3MY
September 11, 2019 at 9:10 am · Reply

#include <iostream>
/* I read the first 2 paragraphs and wanted to see if i could do it myself, so i quickly wrote this silly little program. Wasn't confident at first but it turned out pretty good! *\

int main()
{
  int inP{};
  std::cout << "Enter A Number You Want To Double: ";
    std::cin >> inP;
    std::cout << "\n" << "*CPU powering up* AAAND x2!!!! ~~~ " << (inP * 2);

    return 0;

}

// im sure i can use the \n more effectively tho. i think its time to move on.

Crateer
September 20, 2019 at 3:56 am · Reply

```
//Just move \n over to your String that you got anyways. There is no need to have it
  std::cout << "\n*CPU powering up* AAAND x2!!!! ~~~ " << (inP * 2);
```

Hope that helps

George
August 6, 2019 at 8:40 am · Reply

Using the site to refresh my C++ skills - it's very well organised and written with a lot of relevance. Really liked the following author's note:

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

Very true! :)

Chayim
June 6, 2019 at 11:39 pm · Reply

Why is it need to use 2 output operators << << left ands right of expressions to output? is it because it's and expression and not a statement? and to distinguish and indentidfy expression from statement?

nascardriver
June 7, 2019 at 3:02 am · Reply

You start with a stream object, `std::cout`.
When you call `operator<<` on that stream, it prints whatever you gave it, and returns the stream object.

```
std::cout << "Hello " << "Chayim\n";
// It prints "Hello " using the first `<<`. That `<<` then returns out stream `std::co
// You're left with
std::cout << "Chayim\n";
```

`<<` always applies to the result of the previous `<<` (Unless it's the first).

Chayim
June 7, 2019 at 3:56 am · Reply

It's not clear, the first operator prints out what's written after like << "Hello", why is it needed another << for << "nascardriver\n"?

std::cout << "Double that number is: " << num << '\n'; ------ why does num and '\n' need a seporate << operator??

nascardriver
June 7, 2019 at 4:03 am · Reply

You'll probably understand it once you've gotten a little further into the tutorials.

Alex
June 12, 2019 at 3:11 pm · Reply

Each invocation of operator << can print one thing. num is an object, and '\n' is a character. Two different things, hence two uses of <<.

Zoe
June 3, 2019 at 4:24 am · Reply

Hi. I am using clang on codeblocks with c++14 and when i entered this code:

```
#include <iostream>

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

    int num{ 0 };
    std::cin >> num;

    return 0;
}
```

I get this error :
warning: generalized initializer lists are incompatible with C++98 [-Wc++98-compat].
I have no idea how to get rid of it. Any help would be appreciated.

nascardriver
June 3, 2019 at 6:49 am · Reply

Somewhere in your project settings you should be able to toggle compilers flags. Remove -Wc++98-compat.

Zoe
June 4, 2019 at 7:56 am · Reply

Thanks! - That sorted it - I must have selected it by accident.

Thomas
May 4, 2019 at 3:58 am · Reply

The best solution is not the best solution, but maybe the preferred solution since nothing fancy is going on.
Best practice is to always think before you talk, or you may end up being embarrassed by an error since you are talking without thinking.

ooepewwerew
April 7, 2019 at 2:17 pm · Reply

How do I fix this code?

```
#include <iostream>
#include <stdio.h>
#include <cstdio>

int getNumbers()
{
    std::cout << "Please type any integer: ";
    int x{ 0 };
    std::cin >> x;
    return x;
}

int getOperators()
{
    int mop{ 0 };
    do
    {
        std::cout << "Please type the operator you'd like for your calculation (+, -, *, /
        std::cin >> mop;

        if (mop <= 0 || mop > 4)
        {
            std::cout << "The number you entered isn't one of the options provided. Please
        }

    } while (mop <= 0 || mop > 4);

    return mop;
}

int getOperatorChar()
{
    char ops { ' ' };
    bool cont { true };
    int mop { 0 };

    do
    {
        printf ("Please type the operator you'd like for your calculation (+, -, *, /): ")
        std::cin >> ops;

        if (ops == '+')
            mop = 1;

        else if (ops == '-')
            mop = 2;

        else if (ops == '*')
            mop = 3;

        else if (ops == '/')
            mop = 4;

        else
        {
            std::cout << "The number you entered isn't one of the options provided. Please
        }

    } while (mop <= 0 || mop > 4);

    return mop;
}


int makeDecision()
{
    int yesno{ 0 };
    std::cout << "Would you like to put another integer in your equation? Please answer wi
    std::cin >> yesno;
    return yesno;
}

int makeDecision2()
{
    char dec = { ' ' };
    bool cont = { true };
    int yesno = { 0 };

    do
    {
        printf("Would you like to put another integer in your equation? Please answer with
        std::cin >> dec;

        if (dec == 'Yes')
            yesno = 1;

        else if (dec == 'No')
            yesno = 2;
    }

    while (dec <= 0 || dec > 2);

    {
        std::cout << "The answer you entered isn't part of the options provided. Please in
    }

    return yesno;
}

int makeDecision3()
{
    int yesno{ 0 };
    int answer{ 0 };

    if (yesno == 1)
    return getOperators();

    if (yesno == 2)
        return answer;

    return 0;
}

int getAnswer(int x, int mop, int y)
{
    if (mop == 1)
        return x + y;

    if (mop == 2)
        return x - y;

    if (mop == 3)
        return x * y;

    if (mop == 4)
        return x / y;

    return 0;
}

void printResult(int answer)
{
    std::cout << "The answer to your chosen equation is " << answer << "\n";
}

int main()
{
    std::cout << "Hello! This program is a calculator that allows you to add, subtract, mu

    int num1 { getNumbers() };

    int mop1 = { getOperatorChar() };

    int num2 = { getNumbers() };

    makeDecision();

    makeDecision2();

    makeDecision3();

    int mop2 = { getOperatorChar() };

    int num3 = { getNumbers() };

    int mop3 = { getOperatorChar() };

    int num4 = { getNumbers() };

    int mop4 = { getOperatorChar() };

    int num5 = { getNumbers() };

    int answer = { getAnswer(num1, mop1, num2) };

    int answer2 = { getAnswer(answer, mop2, num3) };

    int answer3 = { getAnswer(answer2, mop3, num4) };

    int answer4 = { getAnswer(answer3, mop4, num5) };

    printResult(answer4);

    return 0;
}
```

VichTrogdor
July 7, 2019 at 7:23 am · Reply

When you are asking for help with code, you need to give the reader more information about what you want to fix. You can't just say, "fix this code." A more helpful statement would be something like, "I just added lines x through y to my code, which is intended to do this, but instead, it is doing this and I'm getting this error code. What error did I make?" Even if what you say isn't all of the information necessary for someone to help you, it at least will be enough that someone will try.

Hunter Coleman
March 28, 2019 at 8:39 pm · Reply

Why is it int num { 0 } instead of int num = 0;?

Is it changes to C++ this year or was the { } an assignator for a flexible variable ready to change?

nascardriver
March 29, 2019 at 2:22 am · Reply

Hi!

This was part of a C++ update in 2011. It's covered in lesson 1.4.

Hunter Coleman
March 29, 2019 at 1:23 pm · Reply

Hello nascardriver,

I reviewed 1.4 and found out why you guys included the { }. Now, it makes more sense. In my previous class of Programming Fundamentals II, my teacher taught copy assignments, and did not emphasize the use of the uniform initialization.

The code process would have been as followed:

```
// Collegiate example
#include <iostream>
using namespace std;

int main()
{
    int num;
    cout << "Enter an integer: ";
    cin >> num;

        num = num * 2;

    cout << "Double that number is: " <<  num * 2 << '\n';

    return 0;
}
```

For the rest of our class, we used copy assignemnts all over. I never knew about the { } except for [] which is assigned to arrays.

I need to continue my path of becoming a VR developer, and in order to do that, I need to update myself with practicing more often.

3.14
March 5, 2019 at 5:52 am · Reply

In the author notes, the word "maintainability" is spelled wrongly

Alex
March 8, 2019 at 1:49 pm · Reply

Fixed. Thanks!

Jorge Guimaraes
February 27, 2019 at 7:42 pm · Reply

Question #1

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

One std::cout assignment can output multiple lines, no need (other than educational) to assign cout multiple times:

```
#include <iostream>

int main() {
   std::cout << "Enter an integer: ";
   int num{ 0 };
   std::cin >> num;
//std::cout appears
   std::cout << "Double of " << num << " is " << num * 2 << ".\nTriple of " << num << " is " << num * 3 << "\n";
```

```
    return 0;
}
```

Gurankas

February 5, 2019 at 5:17 am · Reply

In the last Author's note, second last paragraph, I think you meant to say "readers still find plenty of other things to suggest as improvements!" rather than " readers still plenty of other things to suggest as improvements!". Thanks for these tutorials!

Alex

February 7, 2019 at 6:08 am · Reply

As evidenced. :) Thanks for pointing out the omission.

Iury

February 2, 2019 at 6:21 am · Reply

"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)."

It's said "lines 7 and 8", but it's actually lines 5 and 6, I think this happened because the code that ran into a compile error doesn't have

```
#include <iostream>
```

at the start.

Alex

February 2, 2019 at 8:44 am · Reply

Fixed! Thanks.