1.5 — Introduction to iostream: cout, cin, and endl

BY ALEX ON JANUARY 12TH, 2015 | LAST MODIFIED BY NASCARDRIVER ON JANUARY 12TH, 2020

In this lesson, we'll talk more about std::cout, which we used in our *Hello world!* program to output the text *Hello world!* to the console. We'll also explore how to get input from the user, which we will use to make our programs more interactive.

The iostream library

The **iostream library** is part of the C++ standard library that deals with basic input and output. We'll use the functionality in this library to get input from the keyboard and output data to the console. The *io* part of *iostream* stands for *input/output*.

To use the functionality defined within the *iostream* library, we need to include the *iostream* header at the top of any code file that uses the content defined in *iostream*, like so:

```cpp
#include <iostream>

// rest of code that uses iostream functionality here
```

std::cout

The *iostream* library contains a few predefined variables for us to use. One of the most useful is **std::cout**, which allows us to send data to the console to be printed as text. *cout* stands for "character output".

As a reminder, here's our *Hello world* program:

```cpp
#include <iostream> // for std::cout

int main()
{
    std::cout << "Hello world!"; // print Hello world! to console

    return 0;
}
```

In this program, we have included *iostream* so that we have access to *std::cout*. Inside our *main* function, we use *std::cout*, along with the **insertion operator (<<)**, to send the text *Hello world!* to the console to be printed.

std::cout can not only print text, it can also print numbers:

```cpp
#include <iostream> // for std::cout

int main()
{
    std::cout << 4; // print 4 to console

    return 0;
}
```

This produces the result:

```
4
```

It can also be used to print the value of variables:

```cpp
#include <iostream> // for std::cout

int main()
{
    int x{ 5 }; // define integer variable x, initialized with value 5
    std::cout << x; // print value of x (5) to console
    return 0;
}
```

This produces the result:

```
5
```

To print more than one thing on the same line, the insertion operator (<<) can be used multiple times in a single statement to concatenate (link together) multiple pieces of output. For example:

```cpp
#include <iostream> // for std::cout

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

This program prints:

```
Hello world!
```

Here's another example where we print both text and the value of a variable in the same statement:

```cpp
#include <iostream> // for std::cout

int main()
{
    int x{ 5 };
    std::cout << "x is equal to: " << x;
    return 0;
}
```

This program prints:

```
x is equal to: 5
```

std::endl

What would you expect this program to print?

```cpp
#include <iostream> // for std::cout

int main()
{
    std::cout << "Hi!";
    std::cout << "My name is Alex.";
    return 0;
}
```

You might be surprised at the result:

```
Hi!My name is Alex.
```

Separate output statements don't result in separate lines of output on the console.

If we want to print separate lines of output to the console, we need to tell the console when to move the cursor to the next line.

One way to do that is to use *std::endl*. When output with *std::cout*, *std::endl* prints a newline character to the console (causing the cursor to go to the start of the next line). In this context, *endl* stands for "end line".

For example:

```cpp
#include <iostream> // for std::cout and std::endl

int main()
{
    std::cout << "Hi!" << std::endl; // std::endl will cause the cursor to move to the next lin
    std::cout << "My name is Alex." << std::endl;

    return 0;
}
```

This prints:

```
Hi!
My name is Alex.
```

Tip

In the above program, the second *std::endl* isn't technically necessary, since the program ends immediately afterward. However, it serves two useful purposes: First, it helps indicate that the line of output is a "complete thought". Second, if we later want to add additional output statements, we don't have to modify the existing code. We can just add them.

std::endl vs '\n'

Using std::endl can be a bit inefficient, as it actually does two jobs: it moves the cursor to the next line, and it "flushes" the output (makes sure that it shows up on the screen immediately). When writing text to the console using std::cout, std::cout usually flushes output anyway (and if it doesn't, it usually doesn't matter), so having std::endl flush is rarely important.

Because of this, use of the '\n' character is typically preferred instead. The '\n' character moves the cursor to the next line, but doesn't do the redundant flush, so it performs better. The '\n' character also tends to be easier to read since it's both shorter and can be embedded into existing text.

Here's an example that uses '\n' in two different ways:

```cpp
#include <iostream> // for std::cout

int main()
{
    int x{ 5 };
    std::cout << "x is equal to: " << x << '\n'; // Using '\n' standalone
    std::cout << "And that's all, folks!\n"; // Using '\n' embedded into a double-quoted piece
    return 0;
}
```

This prints:

```
x is equal to: 5
And that's all, folks!
```

Note that when ‘\n’ is used by itself to move the cursor to the next line, the single quotes are needed. When embedded into text that is already double-quoted, the single quotes aren’t needed.

We’ll cover what ‘\n’ is in more detail when we get to the lesson on chars (**4.11 -- Chars**).

Best practice

Prefer ‘\n’ over std::endl when outputting text to the console.

Warning

‘\n’ uses a backslash (as do all special characters in C++), not a forward slash. Using a forward slash (e.g. ‘/n’) instead may result in unexpected behavior.

std::cin

`std::cin` is another predefined variable that is defined in the `iostream` library. Whereas `std::cout` prints data to the console using the insertion operator (<<), `std::cin` (which stands for “character input”) reads input from keyboard using the **extraction operator (>>)**. The input must be stored in a variable to be used.

```
1  #include <iostream>  // for std::cout and std::cin
2
3  int main()
4  {
5      std::cout << "Enter a number: "; // ask user for a number
6      int x{ }; // define variable x to hold user input (and zero-initialize it)
7      std::cin >> x; // get number from keyboard and store it in variable x
8      std::cout << "You entered " << x << '\n';
9      return 0;
10 }
```

Try compiling this program and running it for yourself. When you run the program, line 5 will print “Enter a number: “. When the code gets to line 7, your program will wait for you to enter input. Once you enter a number (and press enter), the number you enter will be assigned to variable *x*. Finally, on line 8, the program will print “You entered ” followed by the number you just entered.

For example (I entered 4):

```
Enter a number: 4
You entered 4
```

This is an easy way to get keyboard input from the user, and we will use it in many of our examples going forward. Note that you don’t need to use ‘\n’ when accepting input, as the user will need to press the *enter* key to have their input accepted, and this will move the cursor to the next line.

If your screen closes immediately after entering a number, please see lesson **0.8 -- A few common C++ problems** for a solution.

Best practice

There’s some debate over whether it’s necessary to initialize a variable immediately before you give it a user provided value via another source (e.g. std::cin), since the user-provided value will just overwrite the initialization value. In line with our previous recommendation that variables should always be initialized, best practice is to initialize the variable first.

We’ll discuss how *std::cin* handles invalid input in a future lesson (**5.10 -- std::cin, extraction, and dealing with invalid text input**).

For advanced readers

The C++ iostream library does not provide a way to accept keyboard input without the user having to press *enter*. If this is something you desire, you’ll have to use a third party library. For console applications, we’d recommend the **pdcurses** library. Many graphical user libraries have their own functions to do this kind of thing.

Summary

New programmers often mix up std::cin, std::cout, the insertion operator (<<) and the extraction operator (>>). Here’s an easy way to remember:

- `std::cin` and `std::cout` always go on the left-hand side of the statement.
- `std::cout` is used to output a value (cout = character output)
- `std::cin` is used to get an input value (cin = character input)
- << is used with std::cout, and shows the direction that data is moving (if `std::cout` represents the console, the output data is moving from the variable to the console). `std::cout << 4` moves the value of 4 to the console
- >> is used with `std::cin`, and shows the direction that data is moving (if std::cin represents the keyboard, the input data is moving from the keyboard to the variable). `std::cin >> x` moves the value the user entered from the keyboard into x

We'll talk more about operators in lesson **1.8 -- Introduction to literals and operators**.

Quiz time

Question #1

Consider the following program that we used above:

```
#include <iostream>  // for std::cout and std::cin

int main()
{
    std::cout << "Enter a number: "; // ask user for a number
    int x{}; // define variable x to hold user input
    std::cin >> x; // get number from keyboard and store it in variable x
    std::cout << "You entered " << x << '\n';
    return 0;
}
```

The program expects you to enter an integer value, as the variable x that the user input will be put into is an integer variable.

Run this program multiple times and describe what happens when you enter the following types of input instead:

a) A letter, such as *h*

Show Solution

b) A number with a fractional component. Try numbers with fractional components less than 0.5 and greater than 0.5 (e.g. *3.2* and *3.7*).

Show Solution

c) A small negative integer, such as *-3*

Show Solution

d) A word, such as *Hello*

Show Solution

e) A really big number (at least 3 billion)

Show Solution

The last suggestion may be particularly surprising. Try it! This happens because x can only hold numbers up to a certain size. After that, it "overflows". We'll discuss overflow in a future lesson.

1.6 -- Uninitialized variables and undefined behavior

Index

1.4 -- Variable assignment and initialization

C++ TUTORIAL | PRINT THIS POST

292 comments to 1.5 — Introduction to iostream: cout, cin, and endl

« Older Comments 1 ... 3 4 5

Vitaliy Sh.
January 11, 2020 at 11:28 pm · Reply

Hi Sires!
(<<), (>>)
** <code>?

"... (and press enter), ..."
"... will need to press the enter key ..."
"... without the user having to press enter."
** Enter (capitalization)?

"In line with our previous recommendation that variables should always be initialized, ..."
** Anchor to 1.4's "Initialize your variables"?

"... we'd recommend the pdcurses library."
** New address: https://pdcurses.org/
** PDCurses (capitalization)?

<code>"std::cout << 4 moves the value ..."</code>
<code>"std::cin >> x moves the value ..."</code>
** ``<code>...</code> moves the value ...`` ?

Omran
January 3, 2020 at 11:59 pm · Reply

i made this program , can someone judge me please or rate my code ??? ,

here it is :

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

```
int main() {

    int x;
    int y;

    std::cin >> x;
    std::cin >> y;
    int sum = x + y;
    if (sum == -x+y or sum == -x-y or sum == x-y ) {

std::cout << "please enter non negative values\n";

}
    else {

std::cout << x <<"+" << y<< "=" <<sum <<endl << "hello world\n";

}


    return 0;
}
```

Vitaliy Shatrov
January 10, 2020 at 6:32 am · Reply

Use the code tag code ... /code (in []).

```
// space omitted
//#include<iostream>
#include <iostream>

// why do your need it there?
//#include <cmath>

// int main() {
// Alex advice is to place the
// beginning brace on the separate line
int main()
{
        // do always initialize the variables upon creation
        int x{};
        int y{};

        // looks fine. Your also can do
        std::cin
            >> x
            >> y
        ;

        // prefer the direct brace initialization
        int sum{ x + y };

        // English:
        // If x = -2 and y = 4;
        //   2 + 4 = 6
        //   2 - 4 = -2
        // -2 - 4 = -6
        // sum is -2 + 4 = 4 -2 = 2
        // below is check for at least one
        //  value being non negative? Why?
        // also use the spaces with operators
        if (        sum == -x + y
                or sum == -x - y
                or sum ==   x - y)
        {
                // usually errors have an '!'
                std::cout << "please, enter non negative values!\n";
        }
        else
        {
                 // not "+", but " + "
                 std::cout
                        << x << " + " << y << " = " << sum
                       // std:: omitted
                        << std::endl << "hello world\n";
        }

        return 0;    // or EXIT_SUCCESS;
}
```

PS: x=-2 y=-4 also prints hello world... if(){} isn't works...

Petertheminer11
December 31, 2019 at 9:51 am · Reply

How do you create an account on this website?

nascardriver
January 1, 2020 at 1:21 am · Reply

You can't create an account. You can change your avatar on gravatar.com, it will be linked to learncpp via your email address.

omran
January 4, 2020 at 12:01 am · Reply

Omran
January 3, 2020 at 11:59 pm · Reply
i made this program , can someone judge me please or rate my code ??? ,

here it is :

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

int main() {

  int x;
  int y;

  std::cin >> x;
  std::cin >> y;
  int sum = x + y;
  if (sum == -x+y or sum == -x-y or sum == x-y) {

std::cout << "please enter non negative values\n";

```
}
   else {
std::cout << x <<"+" << y<< "=" <<sum <<endl << "hello world\n";

}


   return 0;
}
```

nascardriver
January 4, 2020 at 2:48 am · Reply

- Please use code tags when posting code (Yellow message below the reply box).
- Don't use `using namespace` at file-scope.
- Initialize variables with breace initialization.
- Don't use `std::endl` unless you have to. Use '\n' instead.

bill
November 8, 2019 at 8:08 pm · Reply

what does the "int" before main()mean ?

nascardriver
November 9, 2019 at 5:17 am · Reply

It's the type that `main` returns (An integer). Return values are covered in the next chapter.

juan
December 19, 2019 at 5:58 am · Reply

so is it necessary to put "int" before the "main"?

nascardriver
December 19, 2019 at 6:04 am · Reply

Yes (There's also a way to put it behind `main`, but it has to be somewhere).

Helpme
September 20, 2019 at 10:37 pm · Reply

Can anyone help me I always get this error in codeblocks when I put the program

Error: no match for 'operator>>' (operand types are 'std::istream {aka std:: basic_istream<char>}' and 'int() ')

nascardriver
September 21, 2019 at 1:57 am · Reply

Could be a wrong compiler configuration. Try this instead

```
#include <iostream>

int main()
{
    std::cout << "Enter a number: ";
    int x(0);
    std::cin >> x;
    std::cout << "You entered " << x << '\n';
    return 0;
}
```

If it works, you didn't set up your project correctly. Make sure you enabled the highest standard (-std=c++17).

Kyle
December 14, 2019 at 3:10 pm · Reply

I had this same problem, on line 6 instead of: int x(0), use: int x{}

troxodev
September 9, 2019 at 3:36 pm · Reply

Hi! anyone know why when we put in the program:

```
 #include <iostream>  // for std::cout and std::cin

int main()
{
    std::cout << "Enter a number: "; // ask user for a number
    int x{}; // define variable x to hold user input
    std::cin >> x; // get number from keyboard and store it in variable x
    std::cout << "You entered " << x << '\n';
    return 0;
}
```

a big number like 3000000000 get a random number?

nascardriver
September 10, 2019 at 1:21 am · Reply

3000000000 is too big to be stored in an int, so it gets reduced to the largest number possible, which is probably 2147483647 (0x7FFFFFFF).

Mike
September 7, 2019 at 9:04 am · Reply

I'm a little confused regarding the solution to A under the Quiz section.

The solution says "x stays initialized to 0". That infers that because it was originally initiated to a zero-value, that it will "stay' a zero-value if/when a non-numeric value is entered. Yet, I set the value to 5 and tested it again. The result was the same as before: 0. So, it didn't 'stay' initialized to 5, it just defaults to 0.

Alex
September 10, 2019 at 8:46 am · Reply

Yes, as of C++11, you are correct. I've amended the wording. Thanks for pointing that out.

Mike
September 7, 2019 at 8:44 am · Reply

While practicing with variables, I used two variables: x=5, y=2.

For the print statement, I used single quotes to separate the two values with the & symbol, like so:

std::cout << "The two numbers are " << x << '&' << y << '\n';

I quickly realized I needed two additional spaces: one before and one after the & symbol, though I forgot to replace the single quotes with double, and thus got a strange result (521069122).

After some further testing, I see x is the first number and y is the last. The numbers between them seem to be somewhat random.

I'm curious if anyone knows how this is interpreted?

nascardriver
September 8, 2019 at 12:33 am · Reply

If there's more than one character in single quotation marks, it's no longer a `char` but an `int`. The value is implementation-defined.

EyeMayBeIncorrect
October 18, 2019 at 8:02 am · Reply

Leaving out the 5 and 2, which are x and y, 2106912 to hex is 0x202620. 0x20 is ASCII for space, 0x26 is ASCII for &

So, in this implementation, the ' & ' expression got converted to a string concatenation of ASCII codes for space, &, and space, which got treated like an int.

Maverick
August 17, 2019 at 8:07 pm · Reply

Codeblocks uses the statement "using namespace std;" everytime I create a new project. How do I disable this instead of deleting it every time manually?

Fernando
August 17, 2019 at 5:46 pm · Reply

Hey, I checked the next lesson but it doesn't talk about "flush". I have read this https://stackoverflow.com/questions/4751972/endl-and-flushing-the-buffer

But I still don't think I quite understand the need for flushing exactly. I understand that it just says "hey, empty the buffer right now and write the content to the output immediately". But I don't understand WHY we need that.

nascardriver
August 18, 2019 at 1:11 am · Reply

Sometimes the automatic flushing doesn't work as you'd like it to. You might encounter this when you write long-running loops and whatever you printed doesn't appear on screen until the end of the loop. In such cases, it's better to flush manually.

Omar
July 31, 2019 at 7:39 pm · Reply

My question might be a little weird but i'll shoot: It was stated earlier that int x{9.3} will draw an error because 9.3 does not fit into our integer container, but when we declare variable 'int y{}' and use 'std::cin >> y;' we do not get an error. Is this because cin does an implicit conversion from decimal to integer?

nascardriver
August 1, 2019 at 1:41 am · Reply

`std::cin >>` is type-aware. It knows that `y` is an `int` and will only extract an int.
If you enter "9.3", only the "9" will be extracted and ".3" remains in the input stream.

« Older Comments 1 … 3 4 5