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

ALEX AUGUST 30, 2021

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 input/output library

The input/output library (io 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:

```
1 #include <iostream>
2
3 // 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:

```
#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:

```
#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
```

Related content

We discuss what the *std::* prefix actually does in lesson2.8 -- Naming collisions and an introduction to namespaces.

std::endl

What would you expect this program to print?

```
#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:

```
#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 line of the
console
    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:

```
#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 of text
(note: no single quotes when used this way)
    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 theextraction operator (`>>`). The input must be stored in a variable to be used.

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

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 8, 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 10, 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.

Just like it is possible to output more than one bit of text in a single line, it is also possible to input more than one value on a single line:

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

int main()
{
    std::cout << "Enter two numbers separated by a space: ";

    int x{ }; // define variable x to hold user input (and zero-initialize it)
    int y{ }; // define variable y to hold user input (and zero-initialize it)
    std::cin >> x >> y; // get two numbers and store in variable x and y
respectively

    std::cout << "You entered " << x << " and " << y << '\n';

    return 0;
}
```

This produces the output:

```
Enter two numbers separated by a space: 5 6
You entered 5 and 6
```

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 (7.16 -- std::cin and handling invalid input).

For advanced readers

The C++ I/O 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 lesson1.9 -- 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

Next lesson
1.6 Uninitialized variables and undefined behavior

Back to table of contents

Previous lesson
1.4 Variable assignment and initialization

Leave a comment... Put C++ code between triple-backticks (markdown style):```Your C++ coc

Name*

Email*

Avatars from https://gravatar.com/ are connected to your provided email address.

Notify me about replies: POST COMMENT

DP N N FOUT

Newest