




Course

-  ELEC-A7151

 Course materials

 Your points

Code

-  Code Vault



Course Pages

-  MyCourses


-  Teams Channel





This course has already ended.
The latest instance of the course can be found at: [Object oriented programming with C++: 2023 Autumn](#)

« 2 Namespaces

Course materials

4 Strings and vectors »

ELEC-A7151 / Module 1: Basics / 3 I/O streams

3 I/O streams

- You can download the template for the programming tasks of the module as a [zip file](#) from [this link](#).

Contents

- 3 I/O streams
 - 3.1 Using standard input and output
 - 3.2 Programming Tasks

C++ Primer: Chapter 1.2 (A first look at input/output)

Input and output are usually the first things to learn in a programming language, so that a program can interact with a user. Also here we will start with this.

In most systems, input and output are modeled through **stream** abstraction, that itself is language independent, but different programming languages have their own ways of handling the streams. Typically, every program will start with three default streams: standard input, standard output and standard error streams. In addition, a program could open additional streams, for example, to access files in the file system.

In C, the streams were accessed using a set of input and output functions specified in *stdio.h* header. Assuming a familiarity with C, you may recall *printf*, *scanf*, and other calls, with their awkward format specifiers. C++ provides a very different tool for accessing the streams, leveraging C++ features such as overloading (discussed later), making the stream operation easier.

3.1 Using standard input and output

In C++ each stream is an object. The input streams are instances of **istream** type objects, and output streams are instances of **ostream** objects, having slightly different interface to their use. The name of the object representing standard input stream is **cin**, and the name of the object representing standard output stream is **cout**.

C++ accesses the streams using the `<<` and `>>` operators, depending on whether we are handling output to stream (the former), or input from stream (the latter):

```
char c;
std::cin >> c; // read a character from cin
std::cout << c; // write a character to cout
```

You can think of the operators as arrows that show the direction of data movement: `>>` is an arrow to the right, so data is taken from the left and put to the right. Similarly, `<<` is an arrow to the left, so data is taken from the right and put to the left. Remember that the stream you want to read from/write to is always on the left side when using the standard library.

Unlike with C's *printf* or *scanf* functions with explicit format specifiers (`%d`, `%f`, `%s`, and such), the programmer does not need to think about the exact data type when using the operators, but the output format is determined "automatically". Therefore, instead of writing

```
int num = 5;
char* str = "Hello!";
printf("num: %d string: %s\n", num, str);
```

in C++ we just have

```
int num = 5;
std::string str = "Hello!";
std::cout << "num: " << num << " string: " << str << std::endl;
```

The output expression can have a mix of string literals (such as "num: " above) and variables or expressions, that will be concatenated in output. In addition, there can be **manipulators**. One such thing is **std::endl** (used above), which will cause the output to move to the next line (i.e. insert a '\n') and **flush** the stream.

The cout stream is generally buffered. That means that whatever is written to it does not get immediately written to the destination. Instead, the characters are written to a buffer where they wait for the stream to be **flushed** i.e. written to the destination. This improves performance as it is faster to write one big chunk at a time than multiple smaller pieces. It also means that sometimes when you do not see an output which you are expecting, the stream might have not been flushed. This is especially the case with the exercises on this course; if your program is to write to standard output, you must always flush the stream after writing your data or our testing library might not see it. In practise, this means using `std::endl` instead of `\n` or calling `.flush()` before the function returns. Note that in high performance applications you want to prefer the latter method, short explanation for that can be found in the *endl* reference linked below.

The C++ stream mechanism shows its power when we need to output more complex types, such as vectors or structured data types. We will see later how this magic happens, and why it works.

In above example we will see that in C++ we can use a **string** class, instead of a plain char array as in C, and that the names defined in C++ standard library are located in **std** namespace (e.g., *std::cout*).

Similarly, instead of reading an integer from user like this

```
int num;
scanf("%d", &num);
```

in C++ we have

```
int num;
std::cin >> num;
```

Function documentations


Read the documentation for [cout](#), [cin](#), [endl](#) and [flush](#).

3.2 Programming Tasks

Points **5 / 5** My submissions **3**

Deadline Friday, 8 October 2021, 19:59

To be submitted alone

 This course has been archived (Saturday, 17 December 2022, 19:59).

First touch

Objective: Get a first feel of I/O-streams in C++, and test that your programming environment works.

Implement function **Hello()** that outputs

Hello world!

followed by a newline. You should not use the *printf()* as in C, but instead use C++ streams and functions that are defined in C++ "iostream" header.

Instructions on how to run and test your programs locally are available in [Getting Started Module](#).

 **first.cpp**


Choose File No file chosen

Submit

Points **10 / 10** My submissions **4**

Deadline Friday, 8 October 2021, 19:59

To be submitted alone

 This course has been archived (Saturday, 17 December 2022, 19:59).

Rectangle

Objective: Practice use of I/O streams.

Implement function **Rectangle** that asks two values from standard input, for the width and height of the rectangle (in floating point format, `double`), and outputs the area and circumference of the rectangle. Input should look exactly like the following:

Please enter width and height

Output should be:

Area: 7.5
Circumference: 11

User input can be any floating point numbers. You should use the C++ **iostream** functions, not the traditional C-style I/O. In the tests, the user inputs are randomly selected, so you will need to actually calculate the area and circumference.

Instructions on how to run and test your programs locally are available in [Getting Started Module](#).

 **rectangle.cpp**

Choose File No file chosen

Submit

« 2 Namespaces

Course materials

4 Strings and vectors »