28.1 — Input and output (I/O) streams

ALEX[1] SEPTEMBER 11, 2023

Input and output functionality is not defined as part of the core C++ language, but rather is provided through the C++ standard library (and thus resides in the std namespace). In previous lessons, you included the iostream library header and made use of the cin and cout objects to do simple I/O. In this lesson, we'll take a look at the iostream library in more detail.

The iostream library

When you include the iostream header, you gain access to a whole hierarchy of classes responsible for providing I/O functionality (including one class that is actually named iostream). You can find a class hierarchy diagram for the non-file-I/O classes here (https://en.cppreference.com/w/cpp/io)[2].

The first thing you may notice about this hierarchy is that it uses multiple inheritance (that thing we told you to avoid if at all possible). However, the iostream library has been designed and extensively tested in order to avoid any of the typical multiple inheritance problems, so you can use it freely without worrying.

Streams

The second thing you may notice is that the word "stream" is used an awful lot. At its most basic, I/O in C++ is implemented with streams. Abstractly, a **stream** is just a sequence of bytes that can be accessed sequentially. Over time, a stream may produce or consume potentially unlimited amounts of data.

Typically we deal with two different types of streams. **Input streams** are used to hold input from a data producer, such as a keyboard, a file, or a network. For example, the user may press a key on the keyboard while the program is currently not expecting any input. Rather than ignore the users keypress, the data is put into an input stream, where it will wait until the program is ready for it.

Conversely, **output streams** are used to hold output for a particular data consumer, such as a monitor, a file, or a printer. When writing data to an output device, the device may not be ready to accept that data yet -- for example, the printer may still be warming up when the program writes data to its output stream. The data will sit in the output stream until the printer begins consuming it.

Some devices, such as files and networks, are capable of being both input and output sources.

The nice thing about streams is the programmer only has to learn how to interact with the streams in order to read and write data to many different kinds of devices. The details about how the stream interfaces with the actual devices they are hooked up to is left up to the environment or operating system.

Input/output in C++

`ios` is a typedef for `std::basic_ios<char>` that defines a bunch of stuff that is common to both input and output streams. We'll deal with this stuff in a future lesson.

The **istream** class is the primary class used when dealing with input streams. With input streams, the **extraction operator (>>)** is used to remove values from the stream. This makes sense: when the user

presses a key on the keyboard, the key code is placed in an input stream. Your program then extracts the value from the stream so it can be used.

The **ostream** class is the primary class used when dealing with output streams. With output streams, the **insertion operator (<<)** is used to put values in the stream. This also makes sense: you insert your values into the stream, and the data consumer (e.g. monitor) uses them.

The **iostream** class can handle both input and output, allowing bidirectional I/O.

Standard streams in C++

A **standard stream** is a pre-connected stream provided to a computer program by its environment. C++ comes with four predefined standard stream objects that have already been set up for your use. The first three, you have seen before:

1. cin -- an istream object tied to the standard input (typically the keyboard)
2. cout -- an ostream object tied to the standard output (typically the monitor)
3. cerr -- an ostream object tied to the standard error (typically the monitor), providing unbuffered output
4. clog -- an ostream object tied to the standard error (typically the monitor), providing buffered output

Unbuffered output is typically handled immediately, whereas buffered output is typically stored and written out as a block. Because clog isn't used very often, it is often omitted from the list of standard streams.

In the next lesson, we'll take a look at some more I/O related functionality in more detail.

Next lesson

28.2 Input with istream

3

Back to table of contents

4

Previous lesson

27.x Chapter 27 summary and quiz

5

6

B U URL INLINE CODE C++ CODE BLOCK HELP!

Leave a comment...

Name*

Email*

Notify me about replies:

POST COMMENT

Find a mistake? Leave a comment above!

Avatars from https://gravatar.com/[8] are connected to your provided email address.

150 COMMENTS

Newest

imto1

March 13, 2025 11:51 pm PDT

Are streams just buffers?
Streams are provided/maintained by OS?

0 Reply

Nidhi Gupta

February 16, 2025 9:18 pm PST

Error messages are printed to the standard error stream (stderr) using std::cerr, which is short for standard character error. Because it is unbuffered—that is, the output is shown instantly—it is more helpful for debugging and error reporting than std::cout.

Simple Syntax
#include <iostream>

int main() {
std::cerr << "This is an error message!" << std::endl;
return 0;
}

1 Reply

milad khodaveisi

January 2, 2025 11:41 pm PST

thank you very much for this information you share.

3 Reply

Kirill

August 23, 2024 8:01 am PDT

>A standard stream is a pre-connected stream provided to a computer program by its environment.
Could you elaborate, please, what does it mean? Does it mean "instance of a class std::basic_ios" or

something like this?

2 Reply

Alex Author

Reply to Kirill August 24, 2024 1:01 am PDT

It means C++ comes with these 4 standard stream preconfigured for input and output as described.

I believe all of the stream types are derived from std::basic_ios.

1 Reply

Kirill

Reply to Alex August 26, 2024 6:20 am PDT

Thanks. As I understood from cppreference cout and cin are objects of a classes std::ostream and std::istream respectively (not sure if these std::ostream and std::istream are the same classes as std::basic_ostream and std::basic_istream though).

0 Reply

Alex Author

Reply to Kirill August 30, 2024 9:57 am PDT

Yes, per https://en.cppreference.com/w/cpp/io/basic_ostream, `std::ostream` is defined as `std::basic_ostream<char>`, and you can see from the inheritance diagram that `std::basic_ostream` is derived from `std::basic_ios`.

2 Reply

Ahuno

September 10, 2023 11:56 am PDT

So, a stream is a buffer in device controller or something like that?

0 Reply

Alex Author

Reply to Ahuno September 13, 2023 3:48 pm PDT

A stream may use a buffer to hold elements that have arrived but not yet been consumed. Other data structures may be used as well (e.g. a queue).

0 Reply

learnccp lesson reviewer

August 2, 2023 1:27 pm PDT

W LESSON

5 Reply

noctis

July 26, 2023 3:22 am PDT

```
Although the ios class is generally derived from ios_base, ios is typically the most
base class you will be working directly with.
```

The linked source - https://en.cppreference.com/w/cpp/io shows that the `ios` class got renamed to `basic_ios` class

Last edited 1 year ago by noctis

0 Reply

Alex Author

Reply to noctis July 28, 2023 6:56 pm PDT

Actually `ios` is a typedef for `std::basic_ios<char>`. I updated the lesson to be more accurate. Thanks for pointing this out.

1 Reply

Emeka Daniel

May 26, 2023 2:54 pm PDT

Does C programming also utilise the same implementation of using streams to output and input data? Infact do all programming languages utilise this?

If not, then what other implementations are there?

1 Reply

Alex Author

Reply to Emeka Daniel May 30, 2023 3:03 pm PDT

C also uses streams, but a different implementation. Look up `stdout` and `stdin`.

1 Reply

EmtyC

Reply to Alex January 3, 2025 10:54 am PST

Um, isn't it that at the lower level, C++ I/O is connected to `stdout` and `stdin` (may have to do with the C runtime library but am not well indulged in this library, or the low levels of stuff)

0 Reply

Alex Author

Reply to EmtyC January 14, 2025 6:17 pm PST

Yep, std::cout generally routes through stdout, and std::cin through stdin.

So they are generally the same implementation in terms of the underlying mechanism that connects to the OS, but they are not the same implementation in terms of the interface they expose to the developer.

2 Reply

Emeka Daniel

Reply to Alex May 31, 2023 1:56 am PDT

Okay, thanks

0 Reply

Tcorn

March 28, 2023 5:53 am PDT

Is normal variable a data consumer ??? Because if that happpen , then std::cin becomes both input and output stream ,be a output stream when we type something like 9hddh for an integer (hddh is stored for later use). How do you guys think ???

Last edited 2 years ago by Tcorn

0 Reply

Alex Author

Reply to Tcorn March 29, 2023 11:11 pm PDT

The terms producer and consumer are generally used in the context of things that work with streams of data.

Typically we'd say std::cin is the stream, and `operator>>` is the consumer, pulling data out of the input stream and placing it in a variable.

4 Reply

Tcorn

March 28, 2023 2:06 am PDT

Oh , another chapter!!!. I will appreciate this one .

1 Reply

Links

1. https://www.learncpp.com/author/Alex/
2. https://en.cppreference.com/w/cpp/io
3. https://www.learncpp.com/cpp-tutorial/input-with-istream/
4. https://www.learncpp.com/
5. https://www.learncpp.com/cpp-tutorial/chapter-27-summary-and-quiz/
6. https://www.learncpp.com/input-and-output-io-streams/
7. https://www.learncpp.com/cpp-tutorial/return-by-reference-and-return-by-address/
8. https://gravatar.com/
9. https://g.ezoic.net/privacy/learncpp.com