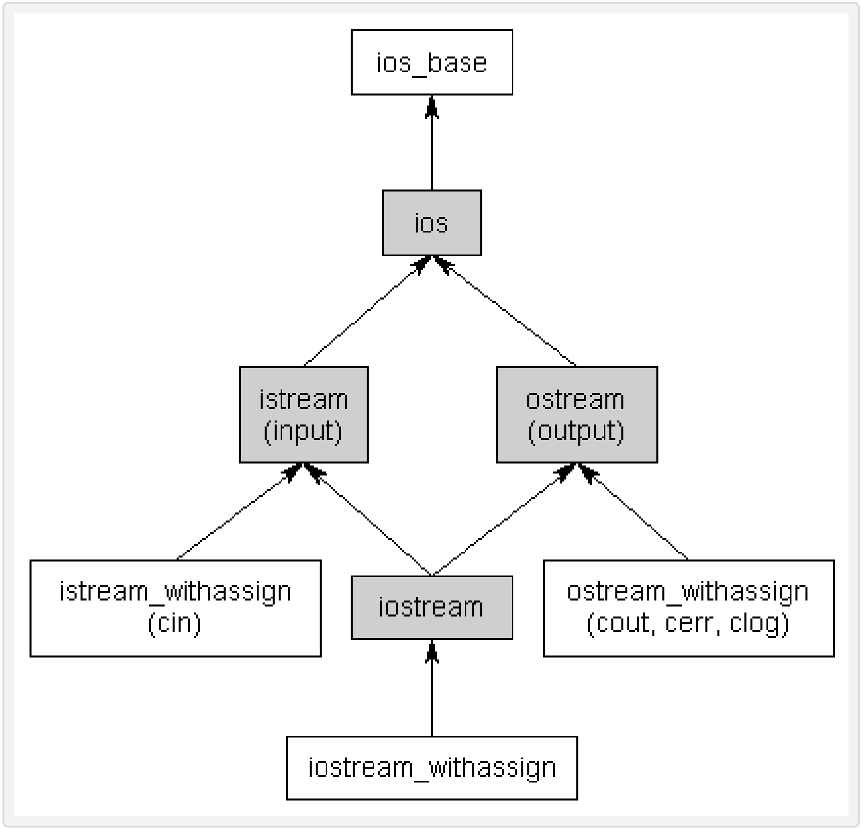
18.1 — Input and output (I/O) streams

BY ALEX ON FEBRUARY 28TH, 2008 | LAST MODIFIED BY ALEX ON NOVEMBER 26TH, 2016

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). The class hierarchy for the non-file-I/O classes looks like this:



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 characters 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++

Although the ios class is generally derived from ios_base, ios is typically the most base class you will be working directly with. The ios class 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 (eg. monitor) uses them.

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

Finally, there are a bunch of classes that end in "_withassign". These stream classes are derived from istream, ostream, and iostream (respectively) with an assignment operator defined, allowing you to assign one stream to another. In most cases, you won't be dealing with these classes directly.

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_withassign class tied to the standard input (typically the keyboard)
2. **cout** -- an ostream_withassign class tied to the standard output (typically the monitor)
3. **cerr** -- an ostream_withassign class tied to the standard error (typically the monitor), providing unbuffered output
4. **clog** -- an ostream_withassign class 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.

A basic example

Here's an example of input and output using the standard streams:

```
#include <iostream>
#include <cstdlib> // for exit()

int main()
{
    using namespace std;
    // First we'll use the insertion operator on cout to print text to the monitor
    cout << "Enter your age: " << endl;

    // Then we'll use the extraction operator on cin to get input from the user
    int nAge;
    cin >> nAge;

    if (nAge <= 0)
    {
        // In this case we'll use the insertion operation on cerr to print an error message
        cerr << "Oops, you entered an invalid age!" << endl;
        exit(1);
    }

    // Otherwise we'll use insertion again on cout to print a result
    cout << "You entered " << nAge << " years old" << endl;

    return 0;
}
```

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