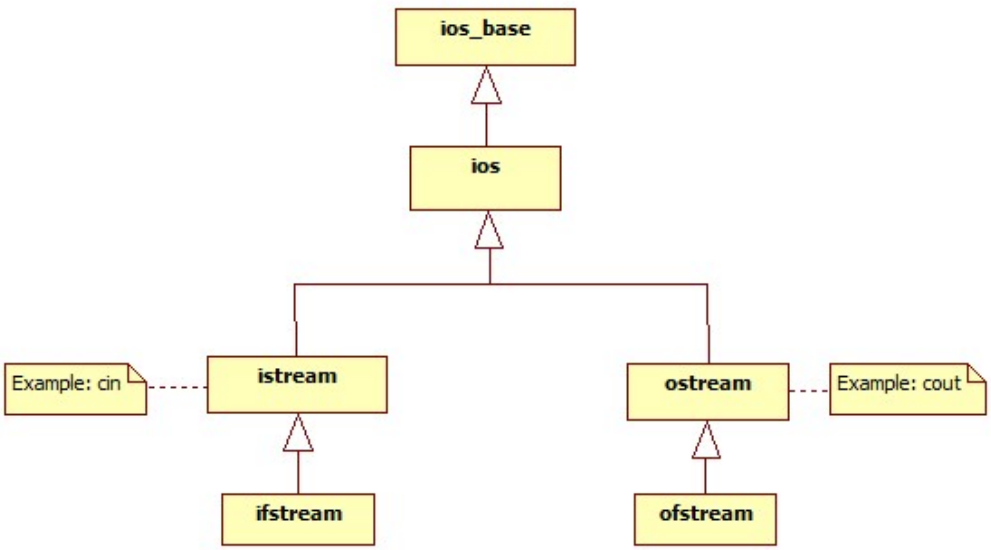
Introduction to C / C++ Programming
File I/O

The Stream Class Hierarchy

- A C++ ***class*** is a collection of data and the methods necessary to control and maintain that data. In this course we will not be writing any C++ classes, but we will learn to use a few, such as ***stream*** classes.
- A class is a data type, analogous to ints, floats, and doubles. A C++ ***object*** is a specific variable having a class as its data type. cin and cout are special pre-specified objects with different classes as their data types.
-
- A C++ ***stream*** is a flow of data into or out of a program, such as the data written to cout or read from cin.
- For this class we are currently interested in four different classes:
  - **istream** is a general purpose input stream. cin is an example of an istream.
  - **ostream** is a general purpose output stream. cout and cerr are both examples of ostreams.
  - **ifstream** is an input file stream. It is a special kind of an istream that reads in data from a data file.
  - **ofstream** is an output file stream. It is a special kind of ostream that writes data out to a data file.
- Object Oriented Programming (e.g. C++) makes heavy use of a concept called ***inheritance,*** in which some classes ***inherit*** the properties of previously written classes. The descendant classes then add on additional properties, making them ***specializations*** of their parent class.
- For example, in the diagram below of (a portion of) the stream class hierarchy, we see that ***ifstream*** is a specialization of ***istream.*** What this means is that an ifstream ***IS*** an istream, and includes all the properties of the istream class, plus some additional properties of its own.



The ifstream Class

- An ***ifstream*** is an ***input file stream***, i.e. a stream of data used for reading input from a file.
- Because an ifstream IS an istream, anything you can do to an istream you can also do the same way to an ifstream.
  - In particular, cin is an example of an istream, so anything that you can do with cin you can also do with any ifstream.
- The use of ifstreams (and ofstreams) requires the inclusion of the fstream header:

```
#include <fstream>
```

- Before you can use an ifstream, however, you must create a variable of type ifstream and connect it to a particular input file.
  - This can be done in a single step, such as:

```
ifstream fin( "inputFile.txt" );
```

  - Or you can create the ifstream and open the file in separate steps:

```
ifstream fin;
fin.open( "inputFile.txt" );
```

- You can even ask the user for the filename, and then open the file they request:

```
string filename;
ifstream fin;
cerr << "Please enter a file name > ";
cin >> filename;
fin.open( filename.c_str( ) ); // The c_str method generates a C-style character string.
```

- (Note that ***string*** is also a class, so in the above example, filename is an object of type string. The string class has a method c_str( ), which we "call" through the specific object using the dot operator, just as we call the open( ) method of the ifstream class..)

- Before you use a newly opened file, you should always check to make sure the file opened properly. Every stream object has a fail( ) method that returns "true" if the stream is in a failed state, or "false" otherwise:

```
if( fin.fail( ) ) {
    cerr << "Error - Failed to open " << filename << endl;
    exit( -1 );  // Or use a loop to ask for a different file name.
}
```

- Once you have created an ifstream and connected it to an open file, you read data out of the file the same way that you read from cin:

```
fin >> xMin;
```

- After you are completely done using a stream,you should always close it to prevent possible corruption.
  - This is especially true for output files, i.e. ofstreams.

```
fin.close( );
```

  - After you have closed a stream, you can re-open it connected to a different file if you wish. (I.e. you can reuse the stream variable.)

The ofstream Class

- An ***ofstream*** is an ***output file stream***, and works exactly like ifstreams, except for output instead of input.
- Once an ofstream is created, opened, and checked for no failures, you use it just like cout:

```
ofstream fout( "outputFile.txt" );
fout << "The minimum oxygen percentage is " << minO2 << endl;
```

Reading Data Files

- One of the key issues when reading input data files is knowing how much data to read, and when to stop reading data. There are three commonly used techniques, as shown below. (Which can also be used when reading from the keyboard.)

I. **Specified Number of Records**
   - One of the easiest ways is to first read in a number indicating how many data items to read in, and then read in that many data items:

```
int nData;
double x, y, z;

fin >> nData;
for( int i = 0; i < nData; i++ ) {
    fin >> x >> y >> z;
    // Do something with x, y, z
} for loop reading input data
```

   - The difficulty with this method is that the number of data items present must be know when the file is created.

II. **Sentinel Value**
   - Another commonly used method is to look for a special value (combination) as a trigger to stop reading data:

```
double x, y;

while( true ) {
    fin >> x >> y;
    if( x = = 0.0 && y = = 0.0 )
        break;
    // Do something with x and y
}  // while loop reading input data
```

- The difficulty with this method is that the sentinel value must be carefully chosen so as not to be possible as valid data.

III. **Detect End of File**

- If you know that the data you are reading goes all the way to the end of the file (i.e. there is no other data in the file after the data you are reading), then you can just keep on reading data until you detect that the end of the file has been reached.
- All istreams (and ifstreams) have a method, eof( ), that returns a boolean value of true AFTER an attempt has been made to read past the end of the file, and false otherwise.
- Because the true value isn't set until AFTER you have gone too far, it is important to: (1) read some data first, then (2) check to see if you've gone past the end of the file, and finally (3) use the data only after you have verified that the reading succeeded. Note carefully in the following code that the check for the end of file always occurs AFTER reading the data and BEFORE using the data that was read:

```
double x, y, z;

fin >> x >> y >> z;
while ( !fin.eof( ) ) {
    // Do something with x, y, z

    // Read in the new data for the next loop iteration.
    fin >> x >> y >> z;
} // while loop reading until the end of file
```

Error Checking Using the Stream State

- Every stream has four bits that keep track of the current ***state*** of the stream:
  - The ***eofbit*** is set to true when an attempt is made to read past the end of the file.
  - The ***badbit*** is set when corrupted data is read, i.e. when the type of data in the file does not match the type being read.
  - The ***failbit*** is set when a file fails to open, or when the end of file is read, or when corrupted data is read.
  - The ***goodbit*** is set to true whenever the other three bits are all false, and is false otherwise.
- The following methods are used to check and reset the bits:
  - eof( ) - returns the state of the eof bit.
  - bad( ) - returns the state of the bad bit.
  - fail( ) - returns the state of the fail bit.
  - good( ) - returns the state of the good bit.
  - clear( ) - Sets the good bit to true and all others to false. This is needed to reset the state if asking the user to enter a new file name after a bad name was entered, or when re-using a stream variable for a new file after encountering the end of a previous file.
- (Advanced) Interestingly, there is an automatic conversion from data type "stream" to "boolean" that can be quite useful.
  - The boolean value generated by the conversion is just the value of the good bit.
    - So if "fin" is an ifstream, then the test "while(fin)" is equivalent to "while(fin.good( ))"
    - So if a stream object is interpreted as a boolean, it will be true if the stream is good and false otherwise.
  - In the following code example, note that the "value" of the >> operator is the stream itself, which is then interpreted as true or false:

```
while( fin >> x >> y >> z ) {

    // Use x, y, and z

    // fin will be "true" until the end of file is read.
    // The reading takes place before the boolean conversion and test.

} // End of while loop reading until the end of file
```