CS 368-1 :: C++ for Java Programmers :: Lecture 11   Please sign the attendance sheet!

A. Preview

Today (Apr 8)	**Today (Apr 15)**	**Next Time(Apr 22)**
• review of operator overloading • overloading == • friends • input/output streams • overloading output <<	• console input/output • example: Point.cpp • overloading cout << • overloading cin >> • error states • file functions • file input/output example • string class, simple member functions, maniplators	• C strings • C I/O • file input/output example

<u>B. Announcements:</u>

1. **Program p4** is due Thursday April 24th at 8:00 PM. **Student Pairs who complete Program p4 using exemplary Pair Programming will each be given 5 bonus points, but may not receive a score above 100 points.** The points may not be carried forward.

2. Notes on p4. This program uses operator overloading to perform arithmetic on Complex Numbers. You can check your answers by entering them into a Google search bar. Also, note that because this file has private data that are only primitives, you do not need to make a destructor.

3. Programs p4 and p5 will lose **10 points per day late**, regardless of the reason. This applies to all situations and circumstances. The purpose of this change is to make late work possible but to discourage it. With 2 programs to finish in 4 weeks, you do not have time to get behind. In order to have your late work graded you must put your work in your “in” directory, and email the instructor as soon as you have completed your late assignment.

C. Input and Output from the keyboard with the Point class. This example will be a good guide for completing program P4.

This class does not have the Big 3, because the copies that are made are good enough for us.

Notice that we can put some basic code into the .h file
(Unlike Java, C++ usually allows you more than one way to do something)

```
//Point.h
//Lecture 11

#ifndef __Lecture11_postLecture__Point__
#define __Lecture11_postLecture__Point__
#include <iostream>
using namespace std;
```

friend allows us to cut some corners here

```
class Point{

    // making this a friend is a short-cut to not writing
    // a separate print() member function
    // do not do this in your Program 4
    friend ostream & operator<< (ostream & out, const Point & p);

    // making this a friend is a short-cut to not
    // calling the Constructor
    // do not do this in your Program 4
    friend istream & operator>> (istream & in, Point & p);
```

note that we have some code in the .h file, which C++ allows

```
public:
    Point () : x(0), y(0) { }
    Point (int a, int b) : x(a), y(b) { }

    // accessors
    int getX() const { return x; }
    int getY() const { return y; }

    // mutators
    void setX(int newX) { x = newX; }
    void setY(int newY) { y = newY; }

private:
    int x, y;
};

#endif /* defined(__Lecture11_postLecture__Point__) */
```

D. Look how nice our main.cpp looks, and notice how well this program runs:

```
// main.cpp
// Lecture 11
#include <iostream>
#include "Point.h"
using namespace std;

int main()
{
    Point p1, p2;

    cout << "enter a point: " ;
    cin  >> p1;
    cout << "you entered: " << p1 << endl;

    cout << "enter a point: " ;
    cin  >> p2;
    cout << "you entered: " << p2 << endl;

    return 0;
}
```

Overloading operator<< was easy, and we used the shortcut of a friend function so that we did not have to write a separate print() function. (For program p4, you will NOT use this short-cut) :

```
// Point.cpp
// because we made this a friend function,
// we can get away with accessing p.x
// in program 4, you will call the member function print()

#include Point.h

ostream & operator<< (ostream & out, const Point & p){
    out << "(" << p.x << "," << p.y << ")";
    return out;  // allows for chaining
}
```

Think about how C++ behaves for primitives and try to copy that behavior

Draw a diagram to show what we will accept.

What will we do if we get bad input that we can't accept?

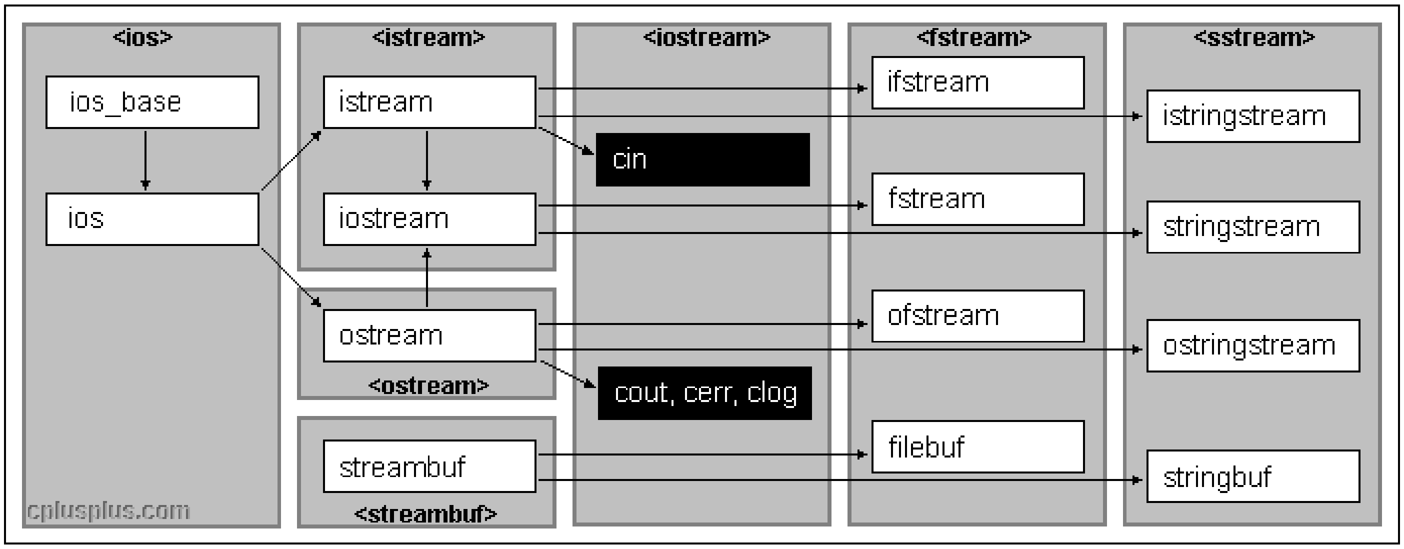
How will we write this code in C++ ?

Here is how we overload input. (In program 4, you will use this method to read over white space. But instead of p.x = x, call the constructor)

```
//  still in Point.cpp

istream & operator>>(istream & in, Point & p) {
    char ch;
    int x, y;
    ch = in.get();  // white space before `(`
    while ( ch == ' ' || ch == '\n' || ch == '\t'){
        ch = in.get();
    }
    if (ch != '('){
        cerr << "missing (" << endl;
        return in;   // not proper input, reject this attempt
    }
    in >> x;      // extract x

    in.get(ch);   // white space before `,'
    while ( ch == ' ' || ch == '\n' || ch == '\t'){
        ch = in.get();
    }
    if (ch != ','){
        cerr << "missing ," << endl;
        return in;   // not proper input, reject this attempt
    }

    in >> y;        // extract y

    in.get(ch);     // white space before `)'
    while ( ch == ' ' || ch == '\n' || ch == '\t'){
        ch = in.get();
    }
    if (ch != ')'){
        cerr << "missing )" << endl;
        return in;   // not proper input, reject this attempt
    }

    // we think we have correct input, so accept it
    p.x = x;   // we could have called the constructor
    p.y = y;

    return in;      // allows for chaining
}
```

E. File Input and Output also use Streams (not used in program P4)

Overview: To create a file stream, you must #include <fstream> After doing so, two new classes, ofstream and ifstream become available to your program.



Notice that all the fstream classes are derived from iostream classes.

There are lots of useful functions that we can use for istreams, here are just a few:

`in.eof()`	returns true if at end of file
`in.get()`	reads and returns one character
`in.unget()`	puts a character back back into the stream
`in.peek()`	returns the next char but keeps it in the stream

If you want to read more about these functions you can look at Cplusplus.com or

F. Using fstream to read in characters and echo them. (not used in program P4)

```
/*
 * fileIO.cpp
 * Updated by Jim Skrentny for CS 368, Fall 2006
 * Written by Beck Hasti for CS 368, Spring 2004
 */

#include <fstream>
#include <iostream>
using namespace std;

int main() {

    ifstream inFile("inFile.txt"); // Open inFile.txt
    if (!inFile) {
        cerr << "Unable to open inFile.txt for reading" << endl;
        return 1;
    }

    ofstream outFile("outFile.txt");  // Open outFile.txt
    if (!outFile) {
        cerr << "Unable to open outFile.txt for writing" << endl;
        return 1;
    }

   //  replace each occurrence
   // of a vowel (a, e, i, o, u) with a *
    char ch;
    while (inFile.get(ch)) { // this handles end of file
        switch (ch) {
          case 'a':  case 'A': case 'e':  case 'E':
          case 'i':  case 'I': case 'o':  case 'O':
          case 'u':  case 'U':
           outFile << '*';   break;
          default:
              outFile << ch;
        }
    }

    inFile.close();
    outFile.close();
    return 0;
}
```

I could not run this program in XCode; you should probably plan to run this program (and programs with file IO) on the CSLab machines.

G. On your own: Reading about Stream Error States….this is background information

Error Checking Using the Stream State (from: http://www.cs.uic.edu/~jbell/CourseNotes/CPlus/FileIO.html)

- 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.

H. C++ String Library Unlike Java, strings in C++ ARE mutable ,
and have overloaded support for == != > >= < <= =

```
//  main.cpp
//  Lecture11_strings
//  Created by Andrew Kuemmel on 4/15/14.

#include <iostream>  // cin, cout
#include <iomanip>   // stream manipulators
#include <string>    // defines string class
using namespace std;

int main()
{
    string s1 = "April?" ;   cout << s1 << endl;
    cout << "the char at pos 3 is " << s1.at(3) << endl;

    // substring is (start_pos, num) ... not like Java
    cout << "s1.substr(2,3) is: " << s1.substr(2,3) << endl;
    cout << "s1.insert(3, \"**\")  returns: " ;
    s1.insert(3,"**"); cout << s1 << endl;

    string s2;
    cout << "\nenter a string: " ;  cin >> s2;

    cout << "you typed: " << s2 << " with a length of: ";
    cout << s2.length() << endl;

    if (s1 == s2)
        cout << s1 << " is the same as " << s2 << endl << endl;
    else if (s1 < s2)
        cout << s1 << " comes before " << s2 << endl << endl;
    else
        cout << s1 << " comes before " << s2 << endl << endl;

    int x = 12345;
    double y = 0.000987654321;
    string s3 = "cs368";

    // cout only shows 6 significant figures of a double
    cout << "x is " << x << ", y is " << y << ", s3 is " << s3 << endl;

    cout << left << setw(10) << x << ", "
         << right << setw(15) << setprecision(6) << fixed << y << ", "
         << s3 << endl;

    return 0;
}
```

You won’t need to know all the functionality of string, but here is a reference if you are interested:
http://www.cplusplus.com/reference/string/string/

I. Formatting Output using string Manipulators