Extra Topic: Overloading >>

The insertion operator >> that we use with **cout** and every other *ostream* is a binary operator.

1. Its first argument is an *ostream*
2. Its second argument is the thing to be printed
3. It outputs the second argument to the the first one
4. It returns its first argument ????

For example if x is an **int** then

cout << x

means output x into stream **cout**

What do we mean when we say that
It returns its first argument

Suppose **int x,y;** The statement

cout << x << y

is the same as

cout << x; cout << y;

The way that it really works is that << is a left-associative operator so the expression is parenthesized as

(cout << x) << y

Since **(cout << x)** returns **cout** this is the same as executing the two statements

cout << x; cout << y;

Complication:
Actually **cout** doesn't return an *ostream*.
It returns a reference to an *ostream* so we must use the & operator.

Example
In ComplexClass we can overload

```
ostream& operator<<(ostream& outs,  ComplexClass A)
{outs << A.Real  << " + " << A.Imag << "*i";
return outs;
}
```

```
//Complex.cpp
//Implementation file for one version of
//ComplexClass

#include <iostream.h>
#include "Complex.h"

ComplexClass::ComplexClass(int x, int y)
{ Real =x; Imag =y;}

ComplexClass::ComplexClass()
{ Real =0; Imag =0;}

int operator==(ComplexClass A, ComplexClass B)
{ return ( (A.Real==B.Real) && (A.Imag==B.Imag) );}

int operator==(ComplexClass A, int B)
{ return ( (A.Real==B) && (A.Imag==0) );}

int operator==(int A, ComplexClass B)
{ return ( (A==B.Real) && (0==B.Imag) );}

ComplexClass operator+(ComplexClass A, ComplexClass B)
{ ComplexClass temp;
  temp.Real = A.Real + B.Real;
  temp.Imag = A.Imag + B.Imag;
  return temp;
    }

ostream& operator<<(ostream& outs,  ComplexClass A)
{outs << A.Real  << " + " << A.Imag << "*i";
return outs;
}
```

```
//Complex.h
//Header file for one version of ComplexClass

class ComplexClass
{
  public:
        //Overload = and +
        friend int operator==(ComplexClass A, ComplexClass B);
        friend int operator==(ComplexClass A, int B);
        friend int operator==(int A, ComplexClass B);
        friend ComplexClass operator+(ComplexClass A, ComplexClass B);

       //Overload <<
       friend ostream& operator<<(ostream& outs,  ComplexClass A);

        //Constructors
        //Initializes number to be x + y*i  or  0+0*i
        ComplexClass(int x, int y);
        ComplexClass();

  private:
        int  Real;
        int  Imag;
};
```

Note: The values returned by a function are usually *passed-by-value*. In C++, it is possible, though, for the values returned to be *passed-by-reference*.

When a function is returning a reference the calling function receives the `lvalue` of the value returned. For example

```
ostream& operator<<(ostream& outs,  ComplexClass A)
{outs << A.Real  << " + " << A.Imag << "*i";
return outs;
}
```

Returning by reference can be tricky. For more information see section 7.4 of Lippman/Lajoie.

```
//CheckCompClass.cpp
//Checks overloading of + and <<
//in ComplexClass

#include <iostream.h>
#include "Complex.h"

int main()
{
      ComplexClass a(1,1), b(1,0);

      cout << "a = " << a << endl << endl;
      cout << "b = " << b << endl << endl ;
      cout << "a + b = "<< a+b <<  endl << endl;
}
***************OUTPUT************
a = 1 + 1*i

b = 1 + 0*i

a + b = 2 + 1*i
```