Section 3:Lecture 9

Introduction

- Functions as Class Members vs. as Friend Functions
- Overloading, <<, >> Overloading

Test Program

```
complex c1, c2, c3;    //declare three complex variables
cin >> c1;             //we can overload the >> operator
cin >> c2;

//test addition
c3 = c1 + c2;          // using overloaded operator +
cout << endl << "c1 + c2 is ";
c3.print(cout);

//test division
c3 = c1 / c2;          // using overloaded operator /
cout << endl << "c1 / c2 is ";
cout << c3;
cout << endl;          //we can overload the << operator
```

Sample Output

- Using the following input:

  4.4 1.5
  3.5 -2.5

- The expected output from our test program will be:

  c1 + c2 is 7.9 + -1i
  c1 / c2 is 0.62973 + 0.878378i

Matrix Addition

Matrix operator+(const Matrix& rhs) const;

Prototype for member function definition.

```
//Member function definition:
Matrix Matrix::operator +(const Matrix& rhs)
const
{
    assert(row == rhs.row && col == rhs.col);
    Matrix temp(rhs);
    for(int i=0; i<row*col; i++)
    {
      temp.pMat[i]+=pMat[i];
    }
    return temp;
}
```

How many times is the *copy* constructor called?

How many times is the destructor called?

```
//Using operator:
Matrix a(4,4), b(4,4), c(4,4);
//…
a = b+c;
a = b.operator+ (c);        //same as above
```

```
Matrix Matrix :: operator ++(){     //prefix
      for(int i=0; i<row*col; i++) {
        ++pMat[I];
     }
     return *this;
}


Matrix Matrix :: operator ++(int){ //postfix
   Matrix temp = *this;
   for(int i=0; i<row*col; i++) {
         ++pMat[i];
    }
    return temp;
}
```

Note: compiler generates the integer argument to force postfix instance to be called.

Overloading << and >> operators

Example:

Matrix m1;

cin >> m1;

cin is the calling object, so >> operator can not be defined as a member function.

Overloading >>

```
friend istream& operator >>(istream&, Matrix&);
        //prototype
```

```
istream& operator >>(istream& in, Matrix& m){
   for(int i=0; i<row*col; i++) {
         in >> pMat[i];
    }
    return in;
}
```

Overloading <<

```
friend ostream& operator <<(ostream&, const
    Matrix&);  //prototype


ostream& operator <<(ostream& out, const Matrix& m){

   for(int i=0; i<row; ++i){
    for(int j=0; j<col; j++) {
          out>> pMat[i*col+j] >> ‘ ‘;
    }
    out << endl;
   }
    return in;
}
```

Error Checking on input operator

- If your input fails because of incorrect format, your function should mark the state of the istream as *bad*

  is.clear(ios::badbit | is.rdstate())

- clear resets entire error state to zero
- clear(ios::badbit) clears all and sets badbit
- is.rdstate() returns the previous state of all bits
- Statement sets the bit vector to the OR of badbit with previous state

Lab Exercise

Add operators:

+

<<

>>

*

to your matrix class.