Section 3:Lecture 8

Operator Overloading

Introduction

- Operators restricted to be overloaded.
- Unary operators
- Binary operators
- Overloading unary operators
- Overloading binary operators.

Operators

- Assignment operator is defined for objects of the same type. Default assignment operator does a bitwise copy.

```
UnitVector v1, v2;
...
v2 = v1;
```

- Other operators are not predefined
  - arithmetic, relational, logical, input and output

Overloading Operators

- Allows class types to be used in the same way that a predefined/built-in data type is used.
- Definitions for operator functions are included in a class definition in the same way as member functions
  - keyword *operator* is part of the name of the function.
  - the name of the function includes one of the predefined C++ operators
- Only predefined operators may be overloaded
- All predefined operators except(`.` `::` `.*` `?:` sizeof) may be overloaded.

Complex Number Class

- A complex number is a number that has two components; the real component and the imaginary component.

  a + bi

- Arithmetic is defined as follows:

(a + bi) + (c + di) = (a + c) + (b + d)i

(a + bi) - (c + di) = (a - c) + (b - d)i

(a + bi) * (c + di) = (ac - bd) + (ad + bc)i

(a + bi) / (c + di) = (ac + bd) / (c2+d**2) +**

[(bc -ad) /(c2+d**2)]i**

Class Declaration

```
class complex
{
    public:
        complex();
        complex(double,double);
        double getReal() const;
        void setReal(double);
        complex operator+(complex) const;
        complex operator-(complex) const;
        complex operator*(complex) const;
        complex operator/(complex) const;
    private:
        double real, imag;
};
```

Implementation - constructors

```
complex::complex():real(0),y(0)
{       //default constructor
}

complex :: complex(double r, double im)
{
        real = r;
        imag = im;
}
```

Implementation – Overloaded Operators

```
complex complex::operator+(complex c) const
{
        complex temp;
        temp.real = real + c.real;
        temp.imag = imag + c.imag;
        return temp;
}
```

Implementation - Continued

```
complex complex::operator/(complex c) const
{
        complex temp;
        temp.real = (real*c.real + imag*c.imag)/
                    ( pow(c.real,2) + pow(imag,2) );
        temp.imag = (imag*c.real - real*c.imag)/
                    ( pow(c.real,2) + pow(imag,2) );
        return temp;
}
```

Practice! – Implement the * operator

(a + bi) * (c + di) = (ac - bd) + (ad + bc)i

```
complex complex::operator*(complex c) const
{
    complex temp;
    temp.real = real*c.real – imag*c.imag;
    temp.imag = real*c.imag + imag*c.real;
    return temp;
}
```

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.

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