Lab 11

Operator Overloading

OBJECTIVE:

Things that will be covered in today's lab:

- Operator Overloading

THEORY:

Operator Overloading provides the ability to use the same operator to perform different actions In C++ the statement c = a + b will compile successfully if a, b and c are of "int" and "float" types and if we attempt to compile the statement when a,b and c are the objects of user-defined classes, the compiler will generate error message but with operator overloading, this can happen.

Operator overloading is done with the help of a special function, called operator function, which defines the operation that the overloaded operator will perform relative to the class upon which it will work. An operator function is created using the keyword *operator*.

Operator functions can be either members or nonmembers of a class. Non-member operator functions are almost always friend functions of the class, however. The way operator functions are written differs between member and nonmember functions.The general format of member operator function is:
Function prototype: (with in a class)

```
return_typeoperator op(arglist)
```

Function definition:

```
return_type class_name :: operator op(arglist)
{
        function body // task defined
}
```

You can also overload an operator for a class by using a nonmember function, which is usually a friend of the class. Since a friend function is not a member of the class, it does not have a *this* pointer. Therefore, an overloaded friend operator function is passed the operands explicitly. This means that a friend function that overloads a binary operator has two parameters, and a friend function that overloads a unary operator has one parameter. When overloading a binary operator using a friend function, the left operand is passed in the first parameter and the right operand is passed in the second parameter. Insertion and extraction operators, Operator function must be a nonmember function of the class. Here is the syntax.

Function prototype :(with in a class)

```
friendostream&operator<<(ostream&, const class_name&);
friendistream&operator>>(istream&, const class_name&);
```

Function definition:

```
ostream&operator<<( ostream& out, const class_name& obj )
{
// ...
return out;
}
istream&operator>>( istream&in, const class_name& obj )
{
// ...
return in;
}
```

We can overload the entire C++ operator except following.

1. Class member access operator (.)
2. Scope resolution operator (::)
3. Size operator (sizeof)
4. Conditional operator (? :)

Example: What should be the output of this program?

```
class Distance
{
private:
int feet, inches;
public:
      Distance(int f, int i) { feet = f; inches = i; }
voidoperator=(const Distance &D )
{
              feet = D.feet;
              inches = D.inches;
      }
void displayDistance() {
              cout <<"F: "<< feet <<" I:"<<  inches << endl;
      }
};
void main()
{
     Distance D1(11, 10), D2(5, 11);
     D1 = D2;
     cout <<"First Distance :";
     D1.displayDistance();
}
```

Exercise 1

You had implemented class for Complex numbers in lab 7. Now, write overloaded functions for the following operators

1. *(Multiplication)
2. +(Addition)
3. –(Subtraction)
4. /(Division)
5. =(Assignment)
6. == (Equality Comparator)

The Class definition is given below:

```
class comp
{
      double real;
      double imag;
public:
      //    default constructor
      //    function that set real and imag part of complex no
      //    function that print complex number
      //    Opertator "+" for addition
      //    Opertator "-" for Subtraction
      //    Opertator "*" for Multiplication
      //    Opertator "/" for Division
      //    Opertator "==" for check both complex are equal or not
      //    Opertator "=" for assignment
};
```

Your program should run for the following main().

```
void main()
{
      comp n1,n2,n3;

      n1.setpara(2,3);
      n2.setpara(1,4);

      n3=n1;
      n3.show();

      (n1+n2).show();

      n3=n1*n2;
      n3.show();

      n3=n1-n2;
```

```
        n3.show();

        n3=n1/n2;
        n3.show();

        if(n1==n2)
                cout<<"Both no have same real and imag part "<<endl;
        else
                cout<<"Unequal !!!"<<endl;

}
```

Exercise 2:

Write a class implementation for a class named **PhoneNumber** giving the specification below:

Data that is associated with this class are:

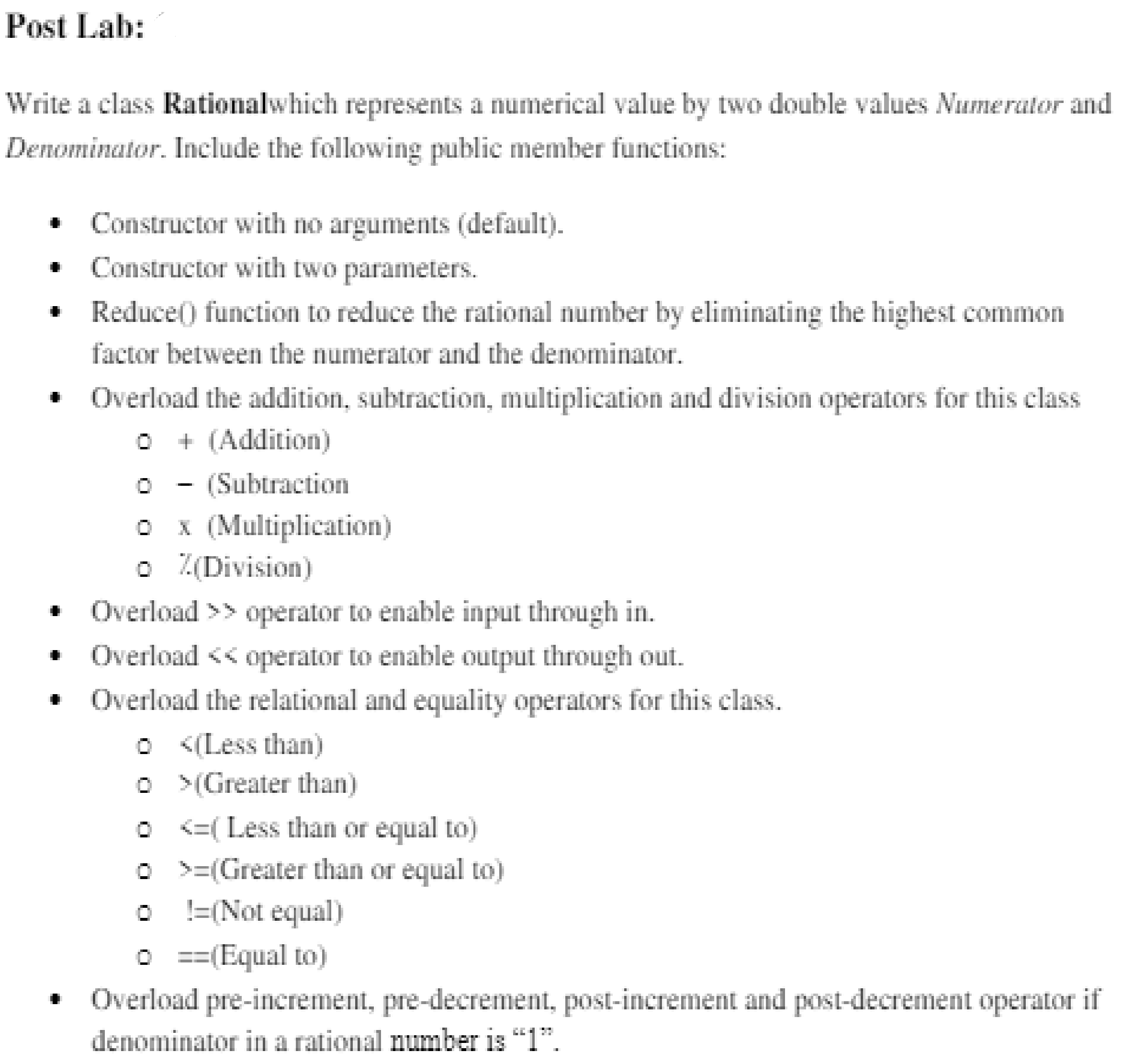
- First name of type string.
- Last name of type string.
- Phone number of type string.

Functions:

- Constructor: that initializes any object of type *"PhoneNumber"*.
- Overloaded function for the insertion operator (<<) to print any object of type *PhoneNumber*.
- Overloaded function for extraction operator (>>) to read in for any object of type *PhoneNumber* all the values of its data members.

Write a driver program that test the class as follow:

- Declare an object of type *"PhoneNumber"*, read in its values, and then print it (using operator<<, operator>>).
- Declare an array of three objects of type *"PhoneNumber"*, read in their values and then print their values (using operator<<, operator>>).



Post Lab:

Write a class **Rational**which represents a numerical value by two double values *Numerator* and *Denominator*. Include the following public member functions:

- Constructor with no arguments (default).
- Constructor with two parameters.
- Reduce() function to reduce the rational number by eliminating the highest common factor between the numerator and the denominator.
- Overload the addition, subtraction, multiplication and division operators for this class
    - + (Addition)
    - − (Subtraction
    - x (Multiplication)
    - %(Division)
- Overload >> operator to enable input through in.
- Overload << operator to enable output through out.
- Overload the relational and equality operators for this class.
    - <(Less than)
    - >(Greater than)
    - <=(Less than or equal to)
    - >=(Greater than or equal to)
    - !=(Not equal)
    - ==(Equal to)
- Overload pre-increment, pre-decrement, post-increment and post-decrement operator if denominator in a rational number is "1".