Lab No.3 Using the global scope resolution operator, passing and returning objects to & from member functions

3.1 Objectives of the lab:

The purpose of this lab is to make you acquainted with the concepts such as

13 Defining member functions outside a class
14 Passing objects in function arguments
15 Returning objects from functions etc.

3.2 Pre-Lab

3.2.1 Member functions defined outside a class

1 Member functions defined inside a class are ***inline*** by default
2 Declaration and definition can be separated from each other
- Declaration inside the class
- Definition outside the class

Syntax:

```
Return_type     class_name  ::   ftn_name   (list of parameters)
{
                Body of function
}
```

1 **::** operator is called ***scope resolution operator***
- specifies what class something is associated with

3.2.2 Objects as function arguments

1 A member function has direct access to all the other members of the class – **public** or **private**
2 A function has indirect access to other objects of the same class that are passed as arguments
- Indirect access: using object name, a dot, and the member name

3.2.3 Example code

```
#include <iostream>
using namespace std;
class complex
{
private:
    double re, im;
public:
    complex();
```

```
    complex(double r, double i);
    void add(complex c1, complex c2);
    void show();
};

complex::complex(): re(0), im(0)
{}
complex::complex(double r, double i): re(r), im(i)
{}

void complex::add(complex c1, complex c2)
{
    re=c1.re + c2.re;
    im=c1.im + c2.im;
}

void complex::show()
{
    cout<<"Complex no: "<<re<<"+"<<im<<"i"<<endl;
}

int main()
{
    complex        c1(3, 4.3), c2(2, 4), c3;
    c3.add(c1, c2);
    c3.show();
    return 0;
}
```

3.2.4 Make changes in the program to allow for a call to add function as c3=c1.add(c2).

3.2.5 Returning objects from functions

Objects can be returned from functions just like normal primitive type variables.

3.2.6 Example code

```
#include <iostream>
using namespace std;
class complex
{
```

```
private:
    double re, im;
public:
    complex();
    complex(double r, double i);
    complex negate();
    void show();
};
complex::complex(): re(0), im(0)
{}
complex::complex(double r, double i): re(r), im(i)
{}

complex complex::negate()
{
    complex temp;
    temp.re= - re;
    temp.im= - im;
    return temp;
}
void complex::show()
{
    cout<<"Complex no: "<<re<<"+"<<im<<"i"<<endl;
}

int main()
{
    complex        c1(3, 4.3), c2(2, 4), c3;
    c3=c1.negate();
    c3.show();
    return 0;
}
```

3.3 In-Lab

3.3.1 Activity

Create a class called **Circle** that has one member data for radius of float type. Provide the following member functions for this class:

a. A **no-argument constructor** to initialize radius to 1.

b. A **1-argument constructor** to initialize the radius to value sent from the calling function at the time

of creation of an object. Make sure that valid values are provided for all the data members. In case of an invalid value, set the variable to 1

c. **showCircle**: displays the radius of circle.

d. **findPerimeter**: calculates and displays the perimeter of the circle.

e. **findArea**: calculates and displays the area of the circle.

f. **isEqual:** should return a variable of type **bool** to indicate whether 1st circle is equal to the 2nd circle or not.

NOTE: Define all the member functions outside the class.

3.3.2 Activity

Create a class called **Time** that has separate int member data for hours, minutes, and seconds. Provide the following member functions for this class:

a. A **no-argument constructor** to initialize hour, minutes, and seconds to 0.

b. A **3-argument constructor** to initialize the members to values sent from the calling function at the time of creation of an object. Make sure that valid values are provided for all the data members. In case of an invalid value, set the variable to 0.

c. A member function **show** to display time in 11:59:59 format.

d. A function **AddTime** for addition of two **Time** objects. Each time unit of one object must add into the corresponding time unit of the other object. Keep in view the fact that minutes and seconds of resultant should not exceed the maximum limit (60). If any of them do exceed, subtract 60 from the corresponding unit and add a 1 to the next higher unit.

NOTE: Define all the member functions outside the class.

A **main()** programs should create two initialized **Time** objects and one that isn't initialized. Then it should add the two initialized values together, leaving the result in the third **Time** variable. Finally it should display the value of this third variable.

3.3.3 Activity

Create a class called **Martix** that represents a 3x3 matrix. This matrix contains a two-dimensional integer array of size 3x3. Provide the following member functions for this class:

a) a **no-argument constructor** for initializing the matrix with 0 values.

b) A **one-argument constructor** to initialize the member matrix to the matrix sent as an argument from the calling function.

c) An **AddMatrix** function for addition of two matrices

d) A **MultiplyMatrix** method for finding the product of the two matrices.

e) An **isEqual** function for checking the equality of two matrices

Note: Define all the member functions outside the class.

3.4 Home-Lab

3.3.4 Activity

Create a class **RationalNumber** that stores a fraction in its original form (i.e. without finding the equivalent floating pointing result). This class models a fraction by using two data members: an integer for numerator and an integer for denominator. For this class, provide the following functions:

a) A **no-argument constructor** that initializes the numerator and denominator of a fraction to some fixed values.

b) A **two-argument constructor** that initializes the numerator and denominator to the values sent from calling function. This constructor should prevent a 0 denominator in a fraction, reduce or simplify fractions that are not in reduced form, and avoid negative denominators.

c) A function **AddFraction** for addition of two rational numbers.

Two fractions a/b and c/d are added together as:

$$\frac{a}{b}+\frac{c}{d}=\frac{(a*d)+(b*c)}{b*d}$$

d) A function **SubtractFraction** for subtraction of two rational numbers.

Two fractions a/b and c/d are subtracted from each other as:

$$\frac{a}{b}-\frac{c}{d}=\frac{(a*d)-(b*c)}{b*d}$$

e) A function **MultiplyFraction** for subtraction of two rational numbers.

Two fractions a/b and c/d are multiplied together as:

$$\frac{a}{b}*\frac{c}{d}=\frac{a*c}{b*d}$$

f) A function **DivideFraction** for division of two rational numbers.

If fraction a/b is divided by the fraction c/d, the result is

$$\frac{a}{b} \Big/ \frac{c}{d} = \frac{a * d}{b * c}$$

g) Provide the following functions for comparison of two fractions

   a. **isGreater**: should return a variable of type **bool** to indicate whether 1st fraction is greater than 2nd or not.

   b. **isSmaller**: should return a variable of type **bool** to indicate whether 1st fraction is smaller than 2nd or not.

   c. **isGreaterEqual**: should return a variable of type **bool** to indicate whether 1st fraction is greater than or equal to 2nd or not.

   d. **isSmallerEqual**: should return a variable of type **bool** to indicate whether 1st fraction is smaller than or equal to 2nd or not.

h) Provide the following functions to check the equality of two fractions

   e. **isEqual**: should return a variable of type **bool** to indicate whether 1st fraction is equal to the 2nd fraction or not.

   f. **isNotEqual:** should a **true** value if both the fractions are not equal and return a **false** if both are equal.

NOTE: Define all the member functions outside the class.

3.3.5 Activity

Create a class called **IntegerSet**. Each object of class **IntegerSet** can hold integers in the range 0 through 49. A set is represented internally as an array of ones and zeros. Array element a[i] is 1 if integer i is in the set. Array element a[j] is 0 if integer j is not in the set. The default constructor initializes a set to the so-called “empty set,” i.e., a set whose array representation contains all zeros.

Provide member functions for the common set operations. For example,

1. Provide a **unionOfIntegerSets** member function that creates a third set which is the set-theoretic union of two existing sets (i.e., an element of the third set’s array is set to 1 if that element is 1 in either or both of the existing sets, and an element of the third set’s array is set to 0 if that element is 0 in each of the existing sets).
2. Provide an **intersectionOfIntegerSets** member function that creates a third set which is the

set-theoretic intersection of two existing sets (i.e., an element of the third set's array is set to 0 if that element is 0 in either or both of the existing sets, and an element of the third set's array is set to 1 if that element is 1 in each of the existing sets).

3. Provide an **insertElement** member function that inserts a new integer k into a set (by setting a[k] to 1).
4. Provide a **deleteElement** member function that deletes integer m (by setting a[m] to 0).
5. Provide a **setPrint** member function that prints a set as a list of numbers separated by spaces. Print only those elements that are present in the set (i.e., their position in the array has a value of 1).
6. Provide an **isEqualTo** member function that determines if two sets are equal.

Now write a driver program to test your **IntegerSet** class. Instantiate several **IntegerSet** objects. Test that all your member functions work properly.

3.5 References:

6 Class notes

7 **Object-Oriented Programming in C++ by *Robert Lafore* (Chapter 6)**

8 **How to Program C++ by *Deitel & Deitel* (Chapter 6)**