Object Oriented Programming

CSE-208(L)

Lab: 03
Using the global scope resolution operator, passing and returning objects to & from member functions

Agenda for Today

- The purpose of this lab is to make you acquainted with the concepts such as:
  - Defining member functions outside a class
  - Passing objects in function arguments
  - Returning objects from functions

✓ **Here we are just doing things other way around**

Defining Member Functions Outside The Class Body

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

Defining Member Functions Outside The Class Body: *Syntax*

```
Return_Type  Class_Name :: Ftn_Name (Parameters_List)

        {
            Body of function
        }
```

Scope Resolution Operator

:: operator is called ***scope resolution operator***

- specifies what class something is associated with

Objects As Function Arguments

- A member function has direct access to all the other members of the class ***public*** or ***private***
- 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***

Quick Example

```
#include <iostream.h>
class complex
{private:
  double re, im;
public:
  complex();
  complex(double r, double i);
  void add(complex c1, complex c2);
  void show();    };
```

Calling members through scope resolution

```
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;  }
```

Passing objects as arguments

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

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

Returning Objects from functions

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

Quick Example

```
#include <iostream.h>
class complex
{
private:
        double      re, im;
public:
        complex();
        complex(double r, double i);
        complex negate();
        void show();        };
```

Returning Object from function

```
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;    }
```

Passing object value

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

Activity No. 01

- 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:

- ✓ **NOTE:** ***Define all the member functions outside the class.***

Activity No. 01 (A)

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

Activity No. 01 (B)

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

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

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

Activity No. 01 (C)

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

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

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

Activity No. 01 (D)

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

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

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

Activity No. 01 (E)

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

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

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

Activity No. 01 (F)

Provide the following functions for comparison of two fractions

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

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

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

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

Activity No. 01 (G)

Provide the following functions to check the equality of two fractions:

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

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