National University of Computer and Emerging Sciences

Lab Manual 07
Object Oriented Programming

Course Instructor	Mr. Usama Hassan
Lab Instructor (s)	Ms. Fariha Maqbool Mr. Sohaib Ahmad
Section	BSE-2C
Semester	Spring 2023

Department of Computer Science
FAST-NU, Lahore, Pakistan

Objectives

After performing this lab, students will practice:

- ✓ Operator Overloading (arithmetic, relational, unary increment (pre/post fix) and stream operators)

TASK 1:

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:

1. A two-argument constructor that initializes the numerator and denominator to the values sent from main. This constructor should prevent a 0 denominator in a fraction, reduce or simplify fractions that are not in reduced form, and avoid negative denominators. If the user skips the values during object creation then it should set the default values of nominator and denominator as 1.
2. An **overloaded >> operator** that takes the nominator and denominator as input from user. This operator should prevent a 0 denominator in a fraction, reduce or simplify fractions that are not in reduced form, and avoid negative denominators.
3. An **overloaded << operator** that displays a fraction in the format a/b.
4. An **overloaded operator +** for addition of two rational numbers. Two fractions a/b and c/d are added together as:

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

5. An **overloaded operator -** for subtraction of two rational numbers. Two fractions a/b and c/d are subtracted from each other as:

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

6. An **overloaded operator ==** that compares the two rational numbers and return true if they are equal and false otherwise
7. An **overloaded prefix operator ++** that increments the values in the nominator and denominator by 1
8. An **overloaded postfix operator –** that decreases the values in the nominator and denominator by 1

```
int main()
{
	RationalNumber RN1;
	RationalNumber RN2(5,6);
	RationalNumber RN3;

	cout<<"Input a rational number"<<endl;
	cin>>RN3;

	cout<<C1<<C2<<C3;

	if(RN1==RN2)
		cout<<"RN1 == RN2"<<endl;
```

```
        else
                cout<<"RN1 != RN2"<<endl;


        RationalNumber RN4= RN2 - RN3;
        cout<<RN4;

        RationalNumber RN5= RN2 + RN3;
        cout<<RN5;

        RationalNumber RN4++;
        cout<<RN4;

        RationalNumber RN5--;
        cout<<RN5;

        system("pause");
        return 0;
}
```

TASK 2:

Remember the 'Set' class which you implemented in the last lab. The data members and operations needed for this ADT are given below.

Private Data Members:

- **int *data;** /* pointer to an array of type integers which will be treated as set */
- **int capacity;** /* maximum possible number of elements that can be stored in the Set */
- **int noOfElements;** /* number of elements in the Set */
- **static int count;** // keep the count of objects of the class created in the main function. Initialize this data member outside the class with 0 and update its value by 1 in the constructors and similarly decrease the value by 1 in the destructor.

The class 'Set' should support the following operations:

1. **Set(int cap = 0);** /* default parameterized constructor */
   Sets 'cap' to 'capacity' and initializes rest of the data members accordingly. If user skips the value in the parameters list then sets the cap to default value. Initially noOfElements in the set were 0 you need to update this value after every insertion in the set.
2. **Set(const Set & ref)** /* copy constructor to implement deep copy */
3. **void reSize (int newcapacity);** /*resize the set to new capacity. Make sure that elements in old set should be preserved in the new set. Since there is a possibility that the newcapacity is either smaller or greater than the original capacity so you need to handle both the cases i.e., shrink and growth. */
4. **static int getObjCount()** /*this function should return the static data member "count" */
5. **~Set()** /* destructor to handle the issue of memory leak and dangling pointer */

Your task is to overload the following overloaded operators:

1- []
   This operator returns an element in the array only if the index is not out of bounds.

2- Pre and Post increment
   Increase value of all elements of array by 1.

3- Pre and Post decrement
   Decrease value of all elements in the array by 1.

4- + Operator
   This operator should increase the values of all elements by an integer **n**

Main function will be as following:

```
int main()
{
        ClassName obj1,
        // Insert the elements in array of obj1 after taking input from user

        cout<<"Enter value of n : ";
        cin>>n;
        ClassName obj2=obj1 + n;       //Use overloaded operator to increment the values of
        elements

        cout<<"After incrementing all element in array by "<<n<<" : ";
        for(int i=0; i<num; i++)
                cout<<obj2[i]<<" ";                  //Get the element using overloaded operator []

        obj1++ ;
        cout<<"After incrementing all element in array by 1 : ";
        for(int i=0; i<num; i++)
                cout<<obj1[i]<<" ";                  //Get the element using overloaded operator []

        obj1--   ;
        cout<<"After incrementing all element in array by 1 : ";
        for(int i=0; i<num; i++)
                cout<<obj1[i]<<" ";                  //Get the element using overloaded operator []

        return 0;

}
```