Object Oriented Programming Lab

FALL - 2022

LAB 10

FAST National University of
Computer and Emerging Sciences

Learning Outcomes

In this lab you are expected to learn the following:

- Operator Overloading

Operator Overloading:

C++ allows you to specify more than one definition for an operator in the same scope, which is called operator overloading. Overloaded operators are functions with special names: the keyword "operator" followed by the symbol for the operator being defined.

Syntax for C++ Operator Overloading:

To overload an operator, we use a special **operator** function. We define the function inside the class or structure whose objects/variables we want the overloaded operator to work with.

```
class className {
    ... .. ...
    public
       returnType operator symbol (arguments) {
           ... .. ...
       }
    ... .. ...
};
```

Here,

- **returnType** is the return type of the function.
- **operator** is a keyword.
- **symbol** is the operator we want to overload. Like: +, <, -, ++, etc.
- **arguments** is the arguments passed to the function.

Example:

```cpp
#include <iostream>
using namespace std;

class Complex {
   private:
    float real;
    float imag;

   public:
    // Constructor to initialize real and imag to 0
    Complex() : real(0), imag(0) {}

    void input() {
        cout << "Enter real and imaginary parts respectively: ";
        cin >> real;
        cin >> imag;
    }

    // Overload the + operator
    Complex operator + (const Complex& obj) {
        Complex temp;
        temp.real = real + obj.real;
        temp.imag = imag + obj.imag;
        return temp;
    }
    void output() {
        if (imag < 0)
            cout << "Output Complex number: " << real << imag << "i";
        else
            cout << "Output Complex number: " << real << "+" << imag << "i";
    }
};

int main() {
    Complex complex1, complex2, result;

    cout << "Enter first complex number:\n";
    complex1.input();

    cout << "Enter second complex number:\n";
    complex2.input();

   // complex1 calls the operator function
   // complex2 is passed as an argument to the function
    result = complex1 + complex2;
    result.output();

    return 0;
}
```

Lab Tasks

Submission Instructions:

1. Create a single cpp file containing all the functions of the problems and main function.
2. Save the **cpp** file with the task number
   e.g. Q1.cpp
3. Now create a new folder with name *ROLLNO_SEC_LAB01* **e.g. i22XXXX_A_LAB10**
4. You need to display your roll no and name before the output of each question.
5. Move all of your .**cpp files (without the main function i.e., comment out the main function)** to this newly created directory and compress it into a **.zip file**.
6. Now you have to submit this zipped file on Google Classroom.
7. If you don't follow the above-mentioned submission instruction, you will be marked **zero.**
8. Plagiarism in the Lab Task will result in **zero** marks in the whole category.

Q1. Write a class **Matrix**.

This class has three private data members:

- rows: An integer that holds the numbers of rows for matrix
- columns: An integer that holds the numbers of columns for matrix
- matrix: An integer pointer to pointer that points to a 2D array (rows * columns).

The class has the following member functions:

Matrix (int r, int c)	Constructs a new Matrix object to represent the given matrix
operator =	Overload = operator to assign values
operator ==	Overload == operator to compare whether matrices are equal or not
M2=M1+1	Overload + operator which takes integer as argument. It preforms scalar addition.
M2=M1-4	Overload - operator which takes integer as argument. It preforms scalar subtraction.
M3=M1+M2	Overload + operator which takes matrix object as argument. It adds two matrixes and returns the result.
M3=M1-M2	Overload - operator which takes matrix object as argument. It subtracts two matrixes and returns the result.

- Write the destructor to deallocate the memory

  ~Matrix()

Q2. Define a class **Item** with the following data members:

- string itemName
- int quantity
- intPrice per unit

1. Write a default constructor

2. Write a parameterized constructor

3. Overload the following operators:

   a. ++ (prefix) operator will increment the quantity of item by 1
   b. - - (prefix) operator will decrement the quantity of item by 1
   c. ++ (postfix) operator will increment the quantity of item by 5
   d. - - (postfix) operator will decrement the quantity of item by 5
   e. += (int n) will increment the quantity of item by n
   f. -=(int n) will decrement the quantity of item by n
   g. + will add quantity of 2 item in the first item only if the name of items are same
   h. ~ will calculate the total price of a single item
   i. ! display the item

Define a class **ShoppingCart** with the following data members:

- Item* list_of_items;
- int TotalPrice
- int currentSize;

4. Overload the following operators:

   a. += (item i) will add an item in the list_of_items,
   b. -= (string s) will remove the item with name s
   c. ~ will calculate the total bill
   d. ! will display the all the items in cart

Create a complete menu for both of the class