Object Oriented Programming Lab

SPRING - 2024

LAB 06

FAST National University of
Computer and Emerging Sciences

Learning Outcomes

In this lab you are expected to learn the following:

- Classes
- Constructors (Default+Parameterized)
- Access Specifiers
- Member Functions

Class:

A class is a user-defined data type that we can use in our program, and it works as an object constructor, or a "blueprint" for creating objects. In other words, a class is a blueprint for the object.
A class is defined in C++ using the keyword **class** followed by the **name** of the class.

```
class className {
   // some data
   // some functions
};
```

For example,

```
class Room {
    public:
        double length;
        double breadth;
        double height;

        double calculateArea(){
            return length * breadth;
        }

        double calculateVolume(){
            return length * breadth * height;
        }

};
```

Here, we defined a class named **Room**.
The variables **length**, **breadth**, and **height** declared inside the class are known as **data members**. And, the functions **calculateArea()** and **calculateVolume()** are known as **member functions** of a class

Constructors:

A constructor is a special type of member function that is called automatically when an object is created.
In C++, a constructor has the same name as that of the class and it does not have a return type.
For example,

```
class  Wall {
  public:
    // create a constructor
    Wall() {
      // code
    }
};
```

Default Constructors:

A constructor with no parameters is known as a default constructor. In the example above, Wall() is a default constructor

```
// C++ program to demonstrate the use of default constructor

#include <iostream>
using namespace std;

// declare a class
class  Wall {
  private:
    double length;

  public:
    // default constructor to initialize variable
    Wall() {
      length = 5.5;
      cout << "Creating a wall." << endl;
      cout << "Length = " << length << endl;
    }
};

int main() {
  Wall wall1;
  return 0;
}
```

Parameterized Constructors:

In C++, a constructor with parameters is known as a parameterized constructor. This is the preferred method to initialize member data.

```cpp
// declare a class
class Wall {
  private:
    double length;
    double height;

  public:
    // parameterized constructor to initialize variables
    Wall(double len, double hgt) {
      length = len;
      height = hgt;
    }

    double calculateArea() {
      return length * height;
    }
};

int main() {
  // create object and initialize data members
  Wall wall1(10.5, 8.6);
  Wall wall2(8.5, 6.3);

  cout << "Area of Wall 1: " << wall1.calculateArea() << endl;
  cout << "Area of Wall 2: " << wall2.calculateArea();

  return 0;
}
```

C++ Access Specifiers:

In C++, there are 3 access modifiers:

1. public
2. private
3. protected (will be covered in the upcoming labs)

1. Public Access Specifiers:

- The public keyword is used to create public members (data and functions).
- The public members are accessible from any part of the program.

2. Private Access Specifiers:

- The private keyword is used to create private members (data and functions).
- The private members can only be accessed from within the class.

Lab Tasks

Submission Instructions:

1. Create a separate .h and .cpp file for each class.
2. Now create a new folder with name *ROLLNO_SEC_LAB06* **e.g. i23XXXX_A_LAB06**
3. Move all of your files to this newly created directory and compress it into a **.zip file**.
4. Now you have to submit this zipped file on Google Classroom.
5. If you don't follow the above-mentioned submission instruction, you will be marked **zero.**
6. Plagiarism in the Lab Task will result in **zero** marks in the whole category.

Q 1. Design a class Complex for handling Complex numbers and include the following private data members:

1. double real
2. double imaginary

The class has the following member functions.

1. A constructor initializing the number with default parameters.
2. Parameterized Constructor.
   - **Complex(double r, double i)**
3. Getters and Setters of the class data members as given below
   - **void setReal(double r)**
   - **double getReal()**
   - **void setImaginary(double i)**
   - **double getImaginary()**
4. Write the following member function in the class
   - **void display()**

     It prints the Complex object in the form a + bi, for example, 3+2i.

   - **Complex addComplex(Complex &c1)**

     It adds both complex numbers and returns the newly generated complex number.

     (a+bi)+(c+di)=(a+c)+(b+d)i

   - **Complex subComplex(Complex &c1)**

     It adds both complex numbers and returns the newly generated complex number.

     (a+bi)-(c+di)=(a-c)+(b-d)i

   - **Complex mulComplex(Complex &c1)**

     It multiplies both complex numbers and returns the newly generated complex number.

(a+bi)(c+di) = (ac−bd) + (ad+bc)i

Output:

1. **Example1:**

   Use Parameterized Constructor to assign the values

   - C1 = 3+2i , C2 = 1+1i
   - Addition = 4+3i
   - Subtraction = 2+1i
   - Multiplication = 1+5i
2. **Example2:**

   Use setters to assign the values

   - C1 = 4-5i , C2 = -7+2i
   - Addition = -3-3i
   - Subtraction = 11-7i
   - Multiplication = -18+43i

Q 2. Write a class called **Line** that represents a line segment between two Points. You need to write a class named **Point** with the following private data members:

- int x
- int y

And the following member functions:

1. Write a Default Constructor
2. Write a Parameterized Constructor
   - **Point(int a, int b)**
3. Create setter and getter methods.
   - **int getx()**
   - **int gety()**
   - **void setx(int a)**
   - **void sety(int b)**
4. Display function to display x and y coordinates of a given Point
   - **void display()**

The Line class will have two private data members of type Point:

- Point P1
- Point P2

Your Line objects should have the following methods:

- **Line(Point &p1, Point &p2)**

  Constructs a new Line that contains the given two Points.

- **Line(int x1, int y1, int x2, int y2)**

  Constructs a new Line that contains the given two Points.

- **Point getP1()**

  Returns this Line's first endpoint.

- **Point getP2()**

  Returns this Line's second endpoint.

- **double getSlope()**

  Returns the slope of this Line. Remember The slope of a line between points (x1, y1) and (x2, y2) is equal to (y2-y1)/(x2-x1).

- **void displayLine()**

  Display the points P1 and P2 of the given line

Output:

1. **Example1:**
   - P1 = (1, 2) , P2= (5, 11)
   - Call all the above functions
   - Slope= 2.25
2. **Example2:**
   - P1 = (3, -4) , P2= (-9, -7)
   - Call all the above functions
   - Slope= 0.25

Q 3. The absence of array bounds checking in C++ is a source of potential hazard. Write a class which will perform bounds checking on integer arrays.

Write a class **IntegerList** with private member variables as:

- int *list

  A pointer to an int . This member points to the dynamically allocated array of integers (which you will be allocating in the constructor).

- int numElements

  An integer that holds the number of elements in the dynamically allocated array.

- int index

  An integer to keep the track record of the index of an array. It would be used to add or remove elements from the array.

And public member functions

1. **IntegerList(int)**

   The class constructor accepts an int argument that is the number of elements to allocate for the array. The array is allocated, and all elements are set to zero.

2. **~IntegerList()**

   Destructor to deallocate memory of list array.

3. **bool isValid(int)**

   This function validates a subscript into the array. It accepts a subscript value as an argument and returns boolean true if the subscript is in the range 0 through numElements - 1. If the value is outside that range, boolean false is returned.

4. **bool addElement(int v)**

   The addElement member function adds a specific element to the array after verifying the index to be valid. It should return true if the element is successfully added to the list else it should return false.

5. **int removeElement()**

   The removeElement member function removes the element at the end of the list and update the index accordingly

6. **int getElement(int)**

   The getElement member function retrieves a value from a specific element in the list array. The argument is the subscript of the element whose value is to be retrieved. The function should use the isValid function to validate the subscript. If the subscript is valid, the value is returned. If the subscript is invalid, the program returns -1.

7. **void displayElements()**

   The displayElements member function displays all the elements of the list.

Output:

1. **Example1:**
   - Create a list of 5 elements
   - Add the following elements to the list
     - 1,7,-6,8,9
   - Now add another element 11 and display the list again
   - Now call removeElement() two times
   - Display the list again
   - Output: 1,7,-6