Object Oriented Programming Lab

FALL - 2022

LAB 07

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 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_LAB07**
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. Build a class **Sale** with private member variables:

- double itemCost; // Cost of the item
- double taxRate; // Sales tax rate

Functionality is mentioned below:

1. Write a default constructor to set the member variable itemCost to 0 and taxRate to 0.
   - **Sale( )**
2. Write a parameterized constructor that accepts the parameter for each member variable such as cost for ItemCost and rate for taxRate
   - **Sale(double cost, double rate)**
3. Generate setters for itemCost and taxRate
   - **void setItemCost(double cost);**
   - **void setTaxRate(double rate);**
4. Write a function **getTax( )** to calculate tax i.e take a product of itemCost and taxRate.
   - **double getTax( )**
5. Write a function **getTotal( )** to calculate the total price of an item i.e. take a sum of itemCost and getTax( ) (calling getTax() will return the calculated tax on item) .
   - **double getTotal( )**

Output:

1. **Example1:**

   Use Parameterized Constructor to assign the values

   - itemCost = 90, taxRate=0.5;
   - Tax: 45

- Total: 135

2. **Example2:**

   Use Setters to assign the values

   - itemCost = 750, taxRate=0.17;
   - Tax: 127.5
   - Total: 877.5

Q3. In this task, you need to implement a class Fraction that has the following data members:

- int num
- int denom

where denom is the denominator value and num is the numerator value.

The class has the following member functions:

1. A constructor initializing the number with default parameters.
   - **Fraction()**

   Note: denominator cannot be zero

2. Parameterized Constructor
   - **Fraction(int n, int d)**

   Note: you have to check for zero as denominator

3. Getters and setters
   - **void setNum(int n)**
   - **void setDenom(int d)**
   - **int getNum()**
   - **int getDenom()**
4. Write the following member functions in the class:
   - **void displayFraction()**

     It displays the fraction in the form num / denom, for example, 2 / 3.

   - **Fraction AddFractions(Fraction& f1)**

     It adds two fractions and returns the resultant fraction.

     Addition: a/b + c/d = (a*d + b*c) / (b*d)

   - **Fraction MultiplyFractions(Fraction& f1)**

     It multiplies two fractions and returns the resultant fraction.

     Multiplication: a/b * c/d = (a*c) / (b*d)

- **Fraction DivideFractions(Fraction& f1)**

  It divides two fractions and returns the resultant fraction.

  Division: a/b / c/d = (a*d) / (b*c)

Output:

1. **Example1:**

   Use Parameterized Constructor to assign the values

   - F1 = 1/2 , F2= 3/4
   - Addition: 10/8
   - Multiplication:3/8
   - Division: 4/6
2. **Example2:**

   Use Setters to assign the values

   - F1 = -2/5, F2= 7/8
   - Addition: 19/40
   - Multiplication: -14/40
   - Division: -16/35

Q 3. Define a class to represent a bank account named as **BankAccount** and a class to represent the **AccountCategory**. The class AccountCategory includes the following members:

Private Data Members:

1. int accountId
2. string accountName

Member functions:

1. Default constructor

   AccountCategory()

2. Parameterized constructor

   AccountCategory(int id, string name)

3. Create setter and getter methods.

   void setAccountId(int id)

   void setAccountName(int name)

int getAccountId()

string getAccountName()

The class **BankAccount** includes the following members:

Private Data Members:

1. depositorName of type string.
2. accountNumber of type string
3. accountCat of type AccountCategory
4. balance of type long

Member functions:

- Default constructor

  BankAccount()

- Parameterized constructor

  BankAccount(string name, string accNum, AccountCategory accCat, long bal)

- To deposit an amount after validating the amount. Return true if deposited successfully else return false.

  bool depositAmount(long amountToDeposit)

- To withdraw an amount after checking the balance.Return False if amount is less than withdrawal amount otherwise return True in case of successful transaction.

  bool withdrawAmount(long amountToWithdraw)

- To get balance in the given accountNum,

  long getAmount()

Output:

1. **Example1:**
   - Create a BankAccount Object with the balance of Rs. 10000
   - Deposit 500 in the Account
   - Then withdraw 20000 from the account
   - Then withdraw 5000 from the account
   - Display the balance
   - Balance: 5500

Q 4. Define a class **Container** for storing integer values, regardless of the sequence and duplication checks. Your class should consist of the following data members:

1. int * values

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

2. int capacity

   An integer that shows total capacity of your container

3. int counter

   A counter variable which increments upon every insertion operation; shows total number of elements inserted yet in container

2. Write the following member functions:

- A default constructor
- A parameterized constructor with single parameter int c; i.e. it initializes the capacity variable, showing the total capacity of the container, and also allocating memory to array. Set the value of the counter to 0.

  Container(int c)

- **bool isFull( )**

  This function checks if the counter has reached to the max capacity of your array

- **void insert(int val)**

  The function requires you to place that item in an array, but before placing the item, do call isFull( ) to check if we have not reached the capacity. Update the counter variable as well.

- **bool search(int val)**

  Provided value has to be searched in an array, note that array is not passed in the parameter list of this function; because it is already accessible i.e. array is a part of this class. Return true if you found the value.

- **void remove( )**

  It removes the value at the end of the array. You need to check the last value with respect to the counter.

- **void display( )**

  This function displays the values of the array.

Output:

1. **Example1:**

- Create a list of 7 elements
- Add the following elements to the list
    - 0, 6, -2, 7, 5, -8, 1
- Now add another element -5 and display the list again
- Now search -2 and 1 in the array
- Now call removeElement() two times
- Display the list again
- Now search 1 in the list
- Final list: 0, 6, -2, 7, 5