EC-200 Data Structures

LAB MANUAL # 01

**Course Instructor:** Lecturer Anum Abdul Salam

**Lab Engineer:** Lab Engineer Ayesha Batool

**Student Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Degree/ Syndicate: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Trait** | **Obtained Marks** | **Maximum Marks** |
| **R1** | **Application Functionality 20%** |  | 20 |
| **R2** | **Specification & Data structure implementation**  **30%** |  | 30 |
| **R3** | **Reusability**  **10%** |  | 10 |
| **R4** | **Input Validation**  **10%** |  | 10 |
| **R5** | **Efficiency**  **20%** |  | 20 |
| **R6** | **Delivery**  **10%** |  | 10 |
| **R7** | **Plagiarism above 80%** |  | 1 |
|  | **Total** |  | 10 |

**Total Marks = O**𝒃𝒕𝒂𝒊𝒏𝒆𝒅 𝑴𝒂𝒓𝒌𝒔 (∑6𝟏 𝑹𝒊 ∗ 𝑹7)

# LAB # 01: Introduction to C++

# Lab Objective:

To practice C++ basics.

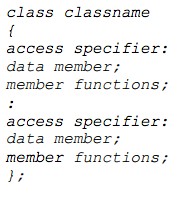
# Lab Description:

## Classes and Objects

Object is the basic unit of object-oriented programming. Object represents a Physical/real entity. All real-world objects have three characteristics:

1. Attributes: The states of an object represent all the information held within it
2. Behavior: Behavior of an object is the set of action that it can perform to change the state of the object.
3. Identity: Object is identified by its unique name

Classes are data types based on which objects are created. Objects with similar attributes and methods are grouped together to form a class. Thus, a class is a logical abstraction, but an object has physical existence. In other words, an object is an instance of a class. Classes are created using the keyword class. A class declaration is similar syntactically to a structure. The attributes of object are represented by variables or data structures (arrays, list etc). The behavior is specified by defining methods, also known as member functions.



Here, access-specifier is one of the three C++ keywords:

* public
* private
* protected

By default, functions and data declared within a class are private to that class and can be accessed only by other members of the class. The public access specifier allows functions or data to be accessible to other parts of your program. The protected access specifier is needed only when inheritance is involved. Once an access specifier has been used, it remains in effect until either another access specifier is encountered or the end of the class declaration is reached.

## Class vs. Struct

The difference between a **class** and a **struct** is simply that a struct's members default to public whereas a class's members default to private. They are otherwise the same.

## C# vs C++

| **Parameter** | **C++** | **C#** |
| --- | --- | --- |
| Type of language | C++ is a low level and platform neutral programming language. | C# is a high-level language. |
| Memory management | In C++, you need to manage memory manually if you dynamically allocate object. | C# runs memory management automatically |
| Multiple inheritances | C++ support the multiple inheritances | C# does not support multiple inheritances. |
| Level of difficulty | C++ includes more complex features. | C# doesn't have any complex features. It has a simple hierarchy and quite easy to understand. |
| Object Oriented | C++ is not a complete object orient language. | C# is a pure object-oriented language. |
| Garbage Collection | C++ does not support garbage collection. | C# supports garbage collection. |
| Multiple inheritance | C++ supports multiple inheritance. | C# does not offer multiple class inheritance. |
| Use of pointers | You can use pointers anywhere in the program. | You can use pointer only in the unsafe mode. |
| Compiler warnings | C++ allows you to do almost anything provided the syntax is right. Therefore, it is flexible language, but you may cause serious damage running OS. | C# is highly protected. as it Compiler will throw errors and warnings in case you inadvertently write code that can cause damage. |
| Switch statement | In C++ Switch Statement, the test variable can't be a string. | In a C# switch statement, may or may not be a string. |

|  |  |
| --- | --- |
| C++ | C# |
| class Car {        // The class   public:          // Access specifier     string brand;  // Attribute     string model;  // Attribute     int year;      // Attribute     Car(string x, string y, int z) { // Constructor with parameters       brand = x;       model = y;       year = z;     } };  int main() {   // Create Car objects and call the constructor with different values   Car carObj1("BMW", "X5", 1999);   Car carObj2("Ford", "Mustang", 1969);    // Print values   cout << carObj1.brand << " " << carObj1.model << " " << carObj1.year << "\n";   cout << carObj2.brand << " " << carObj2.model << " " << carObj2.year << "\n";   return 0; } | using System;class Car{public string brand;public string model;public int year;Car(string x, string y, int z){brand =x;model=y;year=z;}static void Main(string[] args){Car carobj1 = new Car("BMW", "X5",1999);Car carobj2 = new Car("Ford","Mustang",1969);Console.WriteLine("{0} {1} {2}",carobj1.brand,carobj1.model,carobj1.year);Console.WriteLine("{0} {1} {2}",carobj2.brand,carobj2.model,carobj2.year);} |

## Constructors

It is very common for some part of an object to require initialization before it can be used. C++ allows objects to initialize themselves when they are created. This automatic initialization is performed through the use of a constructor function. A constructor is a special function that is a member of a class and has the same name as that class.

## Default constructor

A default constructor is a constructor that can be called without an argument. Note that this includes a constructor that has parameters, provided that all the parameters have been assigned default values. There can only be one of these.

## Parameterized constructor

A parameterized constructor is a constructor which has at least one parameter without a default value. You can create as many of these as you want just like in C#.

## Constructor with Default Arguments

It’s possible to define constructor with default arguments. For e.g. BankAccount( ) can be declared as:



## Destructors

A destructor as name implies, is used to destroy the objects that have been created by a constructor. Like constructor, the destructor is a member function whose name is the same as the class name but is preceded by tilde. For Example, the destructor for the class BankAccount can be defined as shown below:

~BankAccount() { }

A destructor never takes any argument nor does it returns any value. Destructors are used to free memory, release resources and to perform other clean up. Destructors are automatically named when an object is destroyed. If you want to perform some specific task when an object is destroyed, you can define the destructor in the class. If you don’t define, the default one is called.

## Static Variables and Static Class Members

1. A variable declared static within the body of a function maintains its value between invocations of the function.
2. A variable declared static within a module (but outside the body of a function) is accessible by all functions within that module. However, it is not accessible by functions from other modules.
3. Static members exist as members of the class rather than as an instance in each object of the class. There is only a single instance of each static data member for the entire class.
4. Non-static member functions can access all data members of the class: static and non-static. Static member functions can only operate on the static data members.

## Constant

If you make any variable as constant, using const keyword, you cannot change its value. Also, the constant variables must be initialized while they are declared.

int main

{ const int i = 10;

const int j = i + 10; // works fine

i++; // this leads to Compile time error }

In the above code we have made i as constant, hence if we try to change its value, we will get compile time error. Though we can use it for substitution for other variables.

## Example:

|  |
| --- |
| #include <iostream> // using IO functions  #include <string> // using string  using namespace std;  class Circle {  private:  double radius; // Data member (Variable)  string color; // Data member (Variable)  public:  // Constructor with default values for data members  Circle(double r = 1.0, string c = "red") {  radius = r;  color = c;  }  double getRadius() { // Member function (Getter)  return radius;  }  string getColor() { // Member function (Getter)  return color;  }  double getArea() { // Member function  return radius\*radius\*3.1416;  }  }; // need to end the class declaration with a semi-colon    // Test driver function  int main() {  // Construct a Circle instance  Circle c1(1.2, "blue");  cout << "Radius=" << c1.getRadius() << " Area=" << c1.getArea()  << " Color=" << c1.getColor() << endl;    // Construct another Circle instance  Circle c2(3.4); // default color  cout << "Radius=" << c2.getRadius() << " Area=" << c2.getArea()  << " Color=" << c2.getColor() << endl;    // Construct a Circle instance using default no-arg constructor  Circle c3; // default radius and color  cout << "Radius=" << c3.getRadius() << " Area=" << c3.getArea()  << " Color=" << c3.getColor() << endl;  return 0;  } |

# Lab Tasks

## Classes, Constructors and Member Functions

1. Write a class **Square** which has a field for side. It must have a constructor to initialize the side. Add methods to the Square class to calculate area and perimeter.

Declare appropriate member functions const

1. Define a class **Complex\_No** that has two member variables; Real and Imaginary. Also include following in the class:

* A parameterized constructor that takes Real and Imaginary values as argument.
* A default constructor that assigns zero to Real and Imaginary.
* A method Display that shows the value of complex number in appropriate format. i.e., a+bi
* A method Magnitude that calculates the magnitude of complex number
* A method Add that adds two complex numbers and return result; take one complex number as argument.

Declare appropriate member functions const.

**Write a driver program to test your class.** The program should ask for real and imaginary part of two complex numbers, and display the real and imaginary parts of their sum.

1. Define a class **Counter** having an attribute count capable to store int value. provide following functionalities.

The variable counter should keep the track of no of objects being created. Each object should also be capable to hold its instantiated position. Provide following functions to your class

* DisplayTotal () // should be capable to display total number of objects
* MyPos() // should display the number at which a particular object is created

Counter a; // should create an object

Counter b;

a.DisplayTotal(); // 2

a.MyPos();// 1

b.MyPos();// 2

declare appropriate attributes and member functions static and const. Write a driver program that creates multiple objects and then calls the member function to get total number of objects created in memory.