Lab 3: Access Specifiers, Constructors and Destructor

Lab Outcomes:

- Clearly understand the purpose and importance of access specifiers
- Develop a class by correctly using/ enforcing access specifiers
- Differentiate between the *public* and *private* access specifiers
- Understand the importance of constructors
- Understand the importance of destructors
- Use a basic constructor
- Use a basic destructor

22. Introduction

Programming that you studied in the previous semester does not strongly support programming practices like encapsulation and data hiding. In fact these features are almost nonexistent in sequential programming. Object Oriented Programming is purely based on these practices and enforces these practices through various techniques. In this regard access specifiers play a very important role because they are the first line of defense in secure programming. This lab is designed to teach three very basic yet essential concepts of OOP namely access specifiers, constructor and destructors. Access specifiers provide a technique through which we can restrict access to data members and member functions. After enforcing access it is important to reduce our reliance on too many member functions. Constructors offer a very attractive and easy to implement technique through which we can take steps call procedures whenever an object is created. The use of constructors is preferred over the use of member functions because constructors are called automatically and hence they can invoke further functions whereas simple member functions have to be called sequentially and explicitly one by one every time an object is created. Similarly this lab also focuses on the use of destructors that are called whenever an object goes out of scope. In this the lab the students will be familiarized with the use of access specifiers, constructors and destructors.

23. Concept Map

24.1 Access Specifiers – *Public* and *Private* Access

Access specifier also known as access modifier provides a technique for enforcing access control to class members (data members and member functions). The use of access specifiers enforces encapsulation and data hiding. C++ provides three access specifiers i.e. *public*, *private* and *protected*. In this lab we will only cover the *public* and the *private* access specifier. The *protected* specifier is left for future discussions.

The *public* access specifier is the one that provides unrestricted access to the class members. While the *private* access specifier provides a very strict/ restricted access to class members. All the class members that are written under the *public* access can be accessed both inside and outside the class without any restriction. On the other hand all the class members written as *private* are accessible inside the class but are not accessible outside the class. The best and most common way of accessing *private* data members is through the use of a *public* functions.

When we are discussing access specifiers it must be pointed out that by default classes are *private* whereas structures are *public*. Hence if you do not write *private* then your listed class members are considered *private* in a class.

The correct convention for the use of access specifiers is that data members are kept *private* whereas functions are kept *public*. Hence you are providing a public interface for accessing restricted items in a class.

```
#include <iostream>
using namespace std;
class MyClass {
  public:    // Public access specifier
    int x;   // Public attribute
  private:   // Private access specifier
    int y;   // Private attribute
};

int main() {
  MyClass myObj;
  myObj.x = 5;  // Allowed (x is public)
  myObj.y = 5;  // Not allowed (y is private)
  return 0;
```

24.2 Constructors

A constructor is a function that is automatically called when an object is created. This function can exhibit the regular behavior of any function except that it cannot return a value. Constructors are needed because, unlike normal functions that you must call yourself, a constructor runs automatically when you create an object.. Every constructor has a body from where we can call regular member function a very important question which is often asked is that how does the compiler know that the constructor function needs to be called automatically? The answer is very simple. A constructor is a function that has the same name as the class. Whenever an object is created the compiler searches for a function having the same name as the class i.e. the constructor. Given below is a sample code that shows the class constructor. Generally the constructor is defined as *public*. Also the constructor can be overloaded like a regular member function. An important point regarding a constructor is that it cannot return a value. In fact writing the keyword void is strictly prohibited.

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

class myclass
{
private:
   int datamember;   // Private data member

public:
   myclass()   // Class constructor
   {
      cout << "Hello! You have called the class constructor." << endl;
   }
};

int main()
{
   myclass obj;   // Constructor is automatically called here
   return 0;
}
```

Destructors

Constructors are designed to help initialize/ create an object. Destructors on the other hand do exactly the opposite. Destructors are called whenever an object goes out of scope. When this happens it is necessary to perform cleanup procedures especially when you have used dynamic memory or you have been working with pointers in your code. The destructor function can be used to free up memory that you have allocated or dereference pointers that were referenced. The rules for a destructor are as follows:

- They have the same name as the class just simply preceded by a tilde (~)
- They can take no arguments
- They cannot return anything, not even void

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

class myclass
{
private:
   int datamember;   // Private data member

public:
   // Constructor
   myclass()
   {
      cout << "Hello! You have called the class constructor." << endl;
   }

   // Destructor
   ~myclass()
   {
      cout << "Hello! You have called the class destructor." << endl;
   }
};

int main()
{
   myclass obj;  // Constructor called automatically here
   return 0;     // Destructor called automatically when obj goes out of scope
}
```

Lab Tasks

1-From the output write the code

I am default Constructor
I am parameterize Constructor
I am copy Constructor
*********I m dtst
*********I m dtst
*********I m dtst

2- Create a class Book with title and pages.

- Use a parameterized constructor to set values.
- In main(), create an **array of 10 books**, taking input for each one.
- Display all details using a loop. (Book title,Author name,Price,Number of pages, Book ID)

3- Create a class Car with members brand and price.

- Initialize values using a parameterized constructor.
- Create a **copy constructor** that copies data from another object.
- Pass one object to a function displayCar(Car c) (object passed **by value**) and observe when the copy constructor is called automatically.

4- Create:

- Class Car with details like model, rentPerDay, and constructors.
- Class Customer that has a car object, with constructors and destructor.
- A function to calculate total rent (days * rentPerDay)

5-Your task is to carefully understand the code provided. Analyze the code and suggest any corrections that may be needed. There are both syntax and logical errors in the code so consider both when designing the correct solution. Submit the correct code to the lab instructor.

```
class student{
int age;
int cnic;
int semester;
char name;
public:
int setall(int a, int c, int s, int sem, char n) const{
age = a;
cnic = c;
semester = s;
name = n;}}
int myclass :: displayall ( ){
cout<<"The entered data is"<<student.data;}
void main( ){Student obj; obj::setall( ); obj.displayall( ); obj.setage(); Student
anotherobj;Student::anotherobj::setall();
}
```

Lab # 03 Marks distribution

	ER1	ER6	ER8
Task	3 points	3 points	4 points

Lab # 03 Rubric Evaluation Guideline:

#	Qualities & Criteria	0 < Poor <=0.5	0.5 < Satisfactory <= 1.5	1.5 < Excellent <=2
ER 1	**Task Completion**	Minimal or no program functionality was achieved.	Some tasks were completed, but the program has errors or incomplete functionalities.	All tasks were completed, and the program runs without errors.
#	**Qualities & Criteria**	**0 < Poor <=0.5**	**0.5 < Satisfactory <= 1.5**	**1.5 < Excellent <=2**
ER 6	Program Output	Output is inaccurate or poorly presented.	Output is mostly accurate but may lack labels, captions, or formatting.	Output is clear, accurate, and well presented with labels, captions, and proper formatting.
#	**Qualities & Criteria**	**0 < Poor <= 1.5**	**1.5< Satisfactory <= 3**	**3< Excellent <= 4**
ER 8	Question & Answer	Answers some questions but not confidently or based on lab task knowledge.	Answers most questions confidently and based on lab task knowledge.	Answers all questions confidently and demonstrates a deep understanding of the given lab task.