



Lab Manual

**OBJECT ORIENTED PROGRAMMING**

**Semester : Fall 2023**

**Program : BS**

**Course Title and Name : CSC 213**

**Credits : 1**

**Faculty : Mehar Khan Niazi**

**Student Name : Sadia**

**Student ID :20201-28116**

**Total Marks : 100**

**Obtained Marks :**

**Submitted Date :12-oct-2023**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Lab No.** | **Experiment Title** | **Date** |
| **1** | **Transition from C to C++**  **●** Understanding the shift from C to C++  ● Setting up the development environment (IDE,  compiler)  ● Basic syntax and structure transition  ● Hands-on experience with cin, cout, std::  Namespace, variables, loops, and conditions |  |
| **2** | **Overview of programming paradigms/ Introduction to Classes and Objects**  ● Importance of OOP, Why OOP, Procedural  Programing vs OOP  ● Basic syntax and structure of a class in C++  ● Introduction to classes, objects  ● Access Specifiers in Classes |  |
| **3** | **Functions and Constructors**   * Introduction to Constructors * Default Constructors * Passing and Returning Objects to/from Functions * Private Functions and Variable Scopes |  |
| **4** | **Overloading**   * Method Overloading * Constructor Overloading (Parameterized Constructors) * Introduction to Copy Constructors |  |
| **5** | **Classes and Objects Continued**   * Working with Arrays of Objects * Destructors * Constant Member Functions & Objects |  |

**Week 5B Lab**

**Lab Objective:**

The primary objective of this Object-Oriented Programming (OOP) lab session is to provide students with a comprehensive understanding and practical experience in the following key areas of OOP:

**Destructors:**

* Learn the fundamental concept of destructors in object-oriented programming.
* Gain hands-on experience in defining and implementing destructors for user-defined classes.
* Explore the significance of destructors in resource management and object cleanup.

**Constant Member Functions:**

* Understand the crucial role of constant member functions in preserving object integrity and data encapsulation.
* Practice creating and utilizing constant member functions to prevent unintentional modification of object data members.
* Differentiate between constant and non-constant member functions and their appropriate use cases.

**Constant Objects:**

* Explore the concept of constant objects and their role in enforcing immutability.
* Create and manipulate constant objects of user-defined classes.

**Tools/Software Requirement:**

* Dev-C++

**Description Destructors:**

## A destructor is a special member function that works just opposite to constructor, unlike constructors that are used for initializing an object, destructors destroy (or delete) the object.

**Syntax of Destructor**

~class\_name()

{

//Some code

}

Similar to constructor, the destructor name should exactly match with the class name. A destructor declaration should always begin with the tilde(~) symbol as shown in the syntax above.

**When does the destructor get called?**

A destructor is automatically called when:

1) The program finished execution.

2) When a scope (the {} parenthesis) containing local variable ends.

3) When you call the delete operator.

**Destructor rules**

1) Name should begin with tilde sign(~) and must match class name.

2) There cannot be more than one destructor in a class.

3) Unlike constructors that can have parameters, destructors do not allow any parameter.

4) They do not have any return type, just like constructors.

5) When you do not specify any destructor in a class, compiler generates a default destructor and inserts it into your code.

**Code Example 1: Destructor**

#include <iostream>

using namespace std;

class HelloWorld{

public:

//Constructor

HelloWorld(){

cout<<"Constructor is called"<<endl;

}

//Destructor

~HelloWorld(){

cout<<"Destructor is called"<<endl;

}

//Member function

void display(){

cout<<"Hello World!"<<endl;

}

};

int main(){

//Object created

HelloWorld obj;

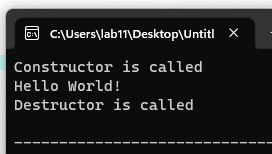
//Member function called

obj.display();

return 0;

}

**Output & Explanation/Reason:**



**Code Examples: Local, Global , Shadowing variable scope**

**Reason :** This is a C++ program that defines a class named HelloWorld. The class has a constructor and a destructor. The constructor is called when an object of the class is created, and the destructor is called when the object is destroyed. The class also has a member function named display that prints “Hello World!” to the console. In the main function, an object of the HelloWorld class is created, and its display function is called.

**Code Example 2: Destructor inside Scope**

#include <iostream>

using namespace std;

class HelloWorld{

public:

//Constructor

HelloWorld(){

cout<<"Constructor is called"<<endl;

}

//Destructor

~HelloWorld(){

cout<<"Destructor is called"<<endl;

}

//Member function

void display(){

cout<<"Hello World!"<<endl;

}

};

int main(){

//Object created

HelloWorld obj;

{

HelloWorld obj2;

//See Output inside and outside scope of destructor

}

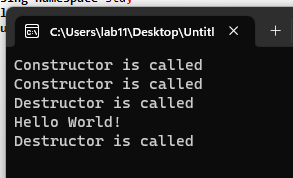
//Member function called

obj.display();

return 0;

}

**Output & Explanation/Reason:**



**Reason :** This is a C++ program that defines a class named HelloWorld. The class has a constructor and a destructor. The constructor is called when an object of the class is created, and the destructor is called when the object is destroyed. The class also has a member function named display that prints “Hello World!” to the console. In the main function, an object of the HelloWorld class is created, and its display function is called. In addition, another object of the same class is created inside a nested scope, which calls its destructor when it goes out of scope**.**

**Code Example 3: Destructor another example**

//The below C++ program demonstrates the number of times constructors and destructors are called.

// C++ program to demonstrate the number of times

// constructor and destructors are called

#include <iostream>

using namespace std;

static int Count = 0; //It is static so that every class object has the same value

class Test {

public:

// User-Defined Constructor

Test()

{

// Number of times constructor is called

Count++;

cout << "No. of Object created: " << Count

<< endl;

}

// User-Defined Destructor

~Test()

{

cout << "No. of Object destroyed: " << Count //It will print count in

<< endl; //decending order

Count--;

// Number of times destructor is called

}

};

// driver code

int main()

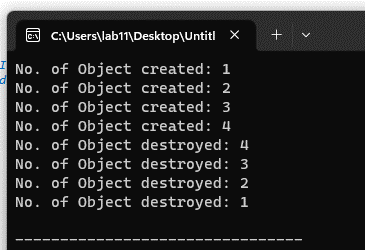
{

Test t, t1, t2, t3;

return 0;

}

**Output & Explanation/Reason:**



**Reason :** This is a C++ program that defines a class named Test. The class has a constructor and a destructor. The constructor increments a static variable Count and prints the number of objects created so far. The destructor decrements Count and prints the number of objects destroyed so far. In the main function, four objects of the Test class are created, which results in the constructor being called four times and the destructor being called four times in reverse order.

**Const member functions and Const Objects**

Constant member functions are those functions that are denied permission to change the values of the data members of their class. To make a member function constant, the keyword const is appended to the function prototype and also to the function definition header.

Like member functions and member function arguments, the objects of a class can also be declared as const. An object declared as const cannot be modified and hence, can invoke only const member functions as these functions ensure not to modify the object. A const object can be created by prefixing the const keyword to the object declaration. Any attempt to change the data member of const objects results in a compile-time error.

**Important Points**

* When a function is declared as const, it can be called on any type of object, const object as well as non-const objects.
* Whenever an object is declared as const, it needs to be initialized at the time of declaration. However, object initialization while declaring is possible only with the help of constructors.
* A function becomes const when the const keyword is used in the function’s declaration. The idea of const functions is not to allow them to modify the object on which they are called.
* It is recommended practice to making as many functions const as possible so that accidental changes to objects are avoided.

**Syntax**

The const member function can be defined as:

**1. For function declaration within a class.**

return\_type function\_name() const;

**Example:**

int get\_data() const;

**2. For function definition within the class declaration.**

return\_type function\_name () const

{

//function body

}

**Example:**

int get\_data() const

{

//function body

}

**Example 4: Examples of Const Member Functions**

The below program demonstrates that data members can be updated in a member function that is not constant.

**Code:**

// C++ program to demonstrate that data members can be

// updated in a member function that is not constant.

#include <iostream>

using namespace std;

class Demo {

int x;

public:

void set\_data(int a) { x = a; }

// non const member function

// data can be updated

int get\_data()

{

++x;

return x;

}

};

main()

{

Demo d;

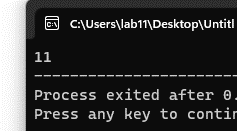
d.set\_data(10);

cout << d.get\_data();

return 0;

}

**Output & Explanation/Reason:**



**Reason** : This is a C++ program that defines a class named Demo. The class has a private integer variable x, a member function named set\_data that sets the value of x, and a member function named get\_data that increments the value of x by 1 and returns it. In the main function, an object of the Demo class is created, its set\_data function is called to set the value of x to 10, and its get\_data function is called to increment the value of x by 1 and print it to the console.

**Example 3: Local , Global , scope using address operator in Function**

**Example 5: Examples of Const Member Functions**

The below C++ program demonstrates that data cannot be updated in a Constant member function.

**Code:**

// C++ program to demonstrate that data cannot be updated

// in a Constant member function

#include <iostream>

using namespace std;

class Demo {

int x;

public:

void set\_data(int a) { x = a; }

// constant member function

int get\_data() const

{

// Error while attempting to modify the data

// member

++x;

return x;

}

};

main()

{

Demo d;

d.set\_data(10);

cout << endl << d.get\_data();

return 0;

}

**Output & Explanation/Reason:**

Error

**Example 6:**

**Reason :** This program demonstrates that data cannot be updated in a constant member function. The program defines a class named Demo with a private integer data member x, and two member functions: set\_data() and get\_data(). The set\_data() function takes an integer argument and sets the value of x to that argument. The get\_data() function is declared as constant using the const keyword, which means that it cannot modify the data members of the class. However, in this case, the get\_data() function attempts to increment the value of x by 1, which is not allowed in a constant member function . In the main() function, an object of the Demo class is created, and its set\_data() function is called to set the value of x to 10. Then, the object’s get\_data() function is called, which attempts to increment the value of x by 1 and returns it. However, since get\_data() is declared as constant, it cannot modify the value of x, so an error occurs

The below C++ program demonstrates that const functions can be called by non-const objects.

**Code:**

// C++ program to demonstrate that const functions can be

// called by non const objects

#include <iostream>

using namespace std;

class Test {

int value;

public:

Test(int v = 0) { value = v; }

// const member function

int getValue() const { return value; }

};

int main()

{

// non const object

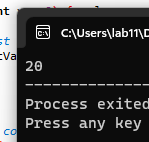
Test t(20);

cout << t.getValue();

return 0;

}

**Output & Explanation/Reason:**



**Reason** : This is a C++ program that defines a class named Test. The class has a constructor that takes an integer argument and initializes a private integer variable value to the argument. The class also has a member function named getValue that returns the value of value. In the main function, an object of the Test class is created with an initial value of 20, and its getValue function is called to print the value of value to the console.

**Example 7: Examples of Const Member Functions**

As we see in example 6 when a function is declared as const, it can be called on any type of object. Non-const functions can only be called non-const objects.

But, the following program has compiler errors.Because when we have const object it can only call const function, not entertaining non-const functions.

**Code:**

// C++ program that demonstrate that non-const functions can

// not be called by const objects

#include <iostream>

using namespace std;

class Test {

int value;

public:

Test(int v = 0) { value = v; }

// non const member function

int getValue() { return value; }

};

int main()

{

// const object

const Test t;

cout << t.getValue();

return 0;

}

**Output & Explanation/Reason:**

ERROR

**Lab Tasks:**

**REASON :** The error in this code is that the getValue function is not a const member function, but it is being called on a const object of the Test class. Since the object is const, it can only call const member functions, which do not modify the object’s data members. To fix this error, you can declare the getValue function as a const member function by adding the const keyword after the function declaration.

**Example 8: Same example as example 7 but in which const function is accessible**

**Code:**

#include <iostream>

using namespace std;

class Demo {

int value;

public:

Demo(int v = 0)

{

value = v;

}

void showMessage()

{

cout << "Hello World showMessage() Function"<< endl;

}

// const member function

void display() const

{

cout << "Hello world Inside display() Function"<< endl;

}

};

int main()

{

// Constant object are initialised at the time of

// declaration using constructor

const Demo d1;

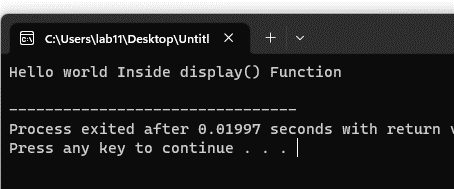
// d1.showMessage();Error occurred if uncomment.

d1.display();

return (0);

}

**Output & Explanation/Reason:**



**Reason :** This is a C++ program that defines a class named Demo. The class has a constructor that takes an integer argument and initializes a private integer variable value to the argument. The class also has two member functions: showMessage and display. The showMessage function prints “Hello World showMessage() Function” to the console, while the display function prints “Hello world Inside display() Function” to the console. In the main function, a constant object of the Demo class is created using the default constructor, and its display function is called. Attempting to call its showMessage function results in an error because it is not a const member function.

**Lab Tasks**

**Requirements (Code, Output and Reason)**

**Task 1: Basic Destructor**

Create a class with a destructor. Instantiate an object of that class and observe when the destructor is called. Explain the order of object destruction.

**Task 2: Const Member Function**

Create a class with a const member function. Explain the concept of const and demonstrate how the const member function can be called.

**Task 3: Const Object**

Create a const object of a class and try to call both const and non-const member functions on it. Explain the restrictions on using member functions with const objects.