# TRIBHUVAN UNIVERSITY

## INSTITUTE OF ENGINEERING

## HIMALAYA COLLEGE OF ENGINEERING

### CHYASAL, LALITPUR

**Lab Report No: - 03**

**Title: - Object ,Classes ,Constructor and Destructor**


**Submitted by: -**

**Name: -Gaurab Pandey**

**Roll NO: - 15**

**Submitted To: -**

**Department Of C and Electronics**

**Checked by:**

**Date of submission: -**

## OBJECTIVE

- To understand the concept of classes and objects in C++.
- To implement constructors for automatic initialization of objects.
- To use destructors for releasing resources and observing object lifecycle.
- To develop modular and reusable code using object-oriented principles.

## BACKGROUND THEORY

In C++, classes are user-defined data types that encapsulate data and functions. An object is an instance of a class, allowing us to represent real-world entities in code. A constructor is a special function that is automatically invoked when an object is created, used to initialize data members. Constructors can be default, parameterized, or copy constructors. A destructor is called automatically when an object goes out of scope or is deleted; it is used to release resources or perform cleanup tasks. Together, these features form the foundation of object- oriented programming, promoting code reusability, abstraction, and maintainability.

**Object:**

An object is any entity, thing or organization that exists in real world that consists of two fundamentals characteristics: its attributes and behavior. In OOP, the problem is divided into a group of objects and each object consists of own properties (data) and behavior (functions).

- Objects are the basic runtime entities which may be created or destroyed at run time.
- Object can communicate with others by using message passing mechanism.
- The member function of an object can only access its data.
- For example; a dog having attributes such as color, weight, age etc. and behaviors such as barking, wagging tail etc.

**Class:**

A class is the collection of similar objects which is defined as the template or prototype to define the common attributes and behavior for all the objects of the class. In fact objects are variables of type class. The entire set of data and code of an object can be made a user-defined data type with the help of a class.

- Once a class has been defined, can create any number of Objects associated with that class.
- No memory is allocated when class is created.
- Class has three access specifiers: public, private and protected. So, class incorporates the concept of data hiding.
- For example, mango, apple and orange are members of class fruit.

**Syntax:**

```
class ClassName {
private:
    // private data members
    // private member functions
protected:
    // protected members (optional)
public:
    // public data members
    // public member functions
};
```

**Example:**

```
#include <iostream>
using namespace std;
class Demo {
private:
    int a;
protected:
    int b;
public:
```

```
    int c;
    void setValues() {
        a = 10;
        b = 20;
        c = 30;
    }
    void display() {
        cout << "Private a = " << a << endl;
        cout << "Protected b = " << b << endl;
        cout << "Public c = " << c << endl;
```

## Constructors:

- In C++, a constructor is a special member function of a class that is automatically called when an object of the class is created.
- It is used to initialize the data members of the class.

**Purpose:**

- Automatically initializes objects when no values are
- passed. Sets default or fixed values to class members.
- Useful for creating multiple objects with the same initial state

**Characteristics:**

1. No parameters.
2. Automatically invoked when an object is created without arguments.
3. Provided by the compiler if no other constructor is defined.
4. Can be explicitly defined by the user.

**Syntax:**

```
class ClassName {
public:
ClassName(); // Default constructor
};
```

**Types of constructors:**

a. Default constructor:

- If you do not define any constructor, the compiler automatically provides a default constructor that does nothing but creates the object.

```
class Test {
// No constructor defined
};
int main() {
Test t1; // Compiler provides default constructor
return 0;
}
```

b. Parameterized constructor:

- A parameterized constructor is a constructor that takes arguments/parameters.
- It is used to initialize objects with specific values at the time of creation.

Syntax:

```
class ClassName {
public:
ClassName(data_type parameter1, data_type parameter2, ...);
};
```

c. Copy constructor:

- A copy constructor is a special constructor in C++ used to create a new object as an exact copy of an existing object.
- It copies the data members of one object to another.

Syntax:

```
class ClassName {
public:
ClassName(const ClassName &obj); // Copy constructor declaration
};
```

## Destructor:

- A destructor is a special member function in C++ that is automatically called when an object goes out of scope or is deleted.
- It is used to free resources allocated to the object.

Purpose:

- To perform clean-up tasks (e.g., releasing memory, closing files).
- To avoid memory leaks in programs using dynamic memory.

## Characteristics:

- Name is the same as the class, prefixed with a tilde ~.
- Takes no arguments and returns nothing.
- Only one destructor is allowed per class (no overloading).
- Automatically invoked at the end of object's lifetime.

**Syntax:**

```
~ClassName() {
// code to release resources
}
```

**Example of constructor and destructor:**

```
#include  <iostream>
using namespace std;
class Demo {
public:
Demo() {
cout << "Constructor called." << endl;
}
~Demo() {
cout << "Destructor called." << endl;
}
void show() {
cout << "Inside show function." << endl;
}
};
```

```
int main() {
Demo obj; // Constructor is called
obj.show(); // Function call
return 0; // Destructor is called automatically
```

1.Define a class Car with private members brand, model, and year. Include public member functions to set and get these private members. Ensure that only member functions can access these private members.

```cpp
#include<iostream>
#include<string>
using namespace std;
class Car {
    string brand;
    string model;
    int year;
    public:
    void setdata()
    {
        cout<<"Brand name: ";
        getline(cin,brand);
        cout<<"Model: ";
        getline(cin,model);
        cout<<"Year: ";
        cin>>year;
    }
    void displaydata()
    {
        cout<<"Brand name: "<<brand<<endl;
        cout<<"Model: "<<model<<endl;
        cout<<"Year: "<<year<<endl;
    }
};
int main()
{
    Car car1;
    cout<<"Enter the details of the car : "<<endl;
    car1.setdata();
    cout<<"Displaying data members using member fucntion. "<<endl;
    car1.displaydata();
    return 0;
```

```
Enter the details of the car :
Brand name: toyota
Model: hilux
Year: 2015
Displaying data members using member fucntion.
Brand name: toyota
Model: hilux
Year: 2015
```

2. Define a class Book with private members title, author, and year. Implement both default and parameterized constructors. The default constructor should initialize the members with default values, and the parameterized constructor should set these values based on user input. Provide a method to display the details of a book.

```
#include<iostream>
#include<string>
using namespace std;
class Book {
    string title;
    string author;
    int year;
    public:
    Book()
    {
        title=" ";
        author=" ";
        year=0;
    }
    Book(string a, string b, int c)
    {
        title=a;
        author=b;
        year=c;
    }
    void display()
    {
        cout<<"Title: "<<title<<endl;
        cout<<"Author: "<<author<<endl;
        cout<<"Year: "<<year<<endl;
    }
};
int main()
{
    Book b1;
    cout<<"Display with default constructor: "<<endl;
    b1.display();
```

```
    Book b2("Fantasy","Michael Zane",2021);
    cout<<"Display with parametetized constructor: "<<endl;
    b2.display();
    return 0;
}
```

```
Display with default constructor:
Title:
Author:
Year: 0
Display with parametetized constructor:
Title: Fantasy
Author: Michael Zane
Year: 2021
```

3.Create a class Employee with private members name and age. Implement a copy constructor to create a deep copy of an existing Employee object. Provide a method to display the details of an employee.

```cpp
#include <iostream>
#include <string>
using namespace std;
class Employee {
private:
    string name;
    int age;
public:
    Employee(string empName, int empAge) {
        name = empName;
        age = empAge;
    }
Employee(const Employee& e) {
        name = e.name;
        age = e.age;
        cout << "Called the copy constructor" << endl;
    }
    void display()  {
        cout << "Employee Name: " << name << endl;
        cout << "Employee Age : " << age << endl;
    }
};
int main() {
    Employee emp1("Hari", 25);
    cout << "Original Details of the Employee:\n";
    emp1.display();
    Employee emp2 = emp1;
    cout << "\nCopied  Details of the Employee:\n";
    emp2.display();
    return 0;
}
```

```
Original Details of the Employee:
Employee Name: Hari
Employee Age : 25
Called the copy constructor

Copied  Details of the Employee:
Employee Name: Hari
Employee Age : 25
```

4.Define a class Book with a private member title. Implement a constructor that initializes title and a destructor that prints a message when the Book object is destroyed. Create instances of Book objects in main to demonstrate the usage of the destructor.

```cpp
#include <iostream>
#include <string>
using namespace std;
class Book {
private:
    string title;

public:

    Book(string t) {
        title = t;
        cout << "Book \"" << title << "\"created." << endl;
    }
    ~Book() {
        cout << "Book \"" << title << "\" destroyed." << endl;
    }
};

int main() {
    cout << "CREATING :" << endl;
    Book book1("Fantasy");

cout << "\n DESTROYING :" << endl;
    return 0;
}
```

```
CREATING :
Book "Fantasy"created.

 DESTROYING :
Book "Fantasy" destroyed.
```

5. Define a class Rectangle with private members length and width. Implement a constructor to initialize these members. Write a function calculateArea() that calculates and returns the area of the rectangle. Define another function doubleDimensions(Rectangle rect) that takes a Rectangle object as an argument and doubles its length and width. Demonstrate the usage of both functions in main().

```cpp
#include <iostream>
using namespace std;
class Rectangle {
private:
    int length;
    int width;

public:
    Rectangle(int l, int w) {
        length = l;
        width = w;
    }
    double calculateArea()  {
        return length * width;
    }
    friend void doubleDimensions(Rectangle &rect);
    void displayDimensions() {
        cout << "Length: " << length << ", Width: " << width << endl;
    }
};
void doubleDimensions(Rectangle &rect) {
    rect.length *= 2;
    rect.width *= 2;
}
    int main() {
    int l, w;
    cout << "Enter the length of the rectangle: ";
    cin >> l;
    cout << "Enter the width of the rectangle: ";
    cin >> w;
     Rectangle myRect(l,w);
    cout << "\n  The original dimensions is:" << endl;
```

```
    cout << "\n  The original dimensions is:" << endl;
    myRect.displayDimensions();
    cout << " Area of original length and width is : " << myRect.calculateArea() << endl;
    doubleDimension (const char [29])"\n After doubling dimensions:"
 cout << "\n After doubling dimensions:" << endl;
    myRect.displayDimensions();
    cout << "The new area is: " << myRect.calculateArea() << endl;
return 0;
}
```

```
Enter the length of the rectangle: 6
Enter the width of the rectangle: 9

  The original dimensions is:
Length: 6, Width: 9
 Area of original length and width is : 54

 After doubling dimensions:
Length: 12, Width: 18
The new area is: 216
```

6. Define a class Student with private members name and age. Implement a constructor to initialize these members. Create an array studentArray of size 3 to store objects of type Student. Populate the array with student details and display the details using a method displayStudentDetails().

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

class Student {
private:
    string name;
    int age;

public:
    Student(string n, int a) {
        name = n;
        age = a;
    }
    Student() {
        name = "";
        age = 0;
    }
void inputStudentDetails() {
        cout << "Enter name: ";
        cin.ignore();
        getline(cin, name);
        cout << "Enter age: ";
        cin >> age;
    }
    void displayStudentDetails() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};
```

```cpp
int main() {
    const int SIZE = 3;
    Student studentArray[SIZE];
    cout << "Enter details for " << SIZE << " students:\n";
    for (int i = 0; i < SIZE; ++i) {
        cout << "\nStudent " << i + 1 << ":\n";
        studentArray[i].inputStudentDetails();
    }
    cout << "\nStudent Details:\n";
    for (int i = 0; i < SIZE; ++i) {
        cout << "Student " << i + 1 << ": ";
        studentArray[i].displayStudentDetails();
    }

    return 0;
}
```

```
Enter details for 3 students:

Student 1:
Enter name: sita
Enter age: 19

Student 2:
Enter name: hari
Enter age: 19

Student 3:
Enter name: ram
Enter age: 20

Student Details:
Student 1: Name: sita, Age: 19
Student 2: Name: hari, Age: 19
Student 3: Name: ram, Age: 20
```

7. Define a class Math with two overloaded methods add(). The first method should take two integers as parameters and return their sum. The second method should take three integers as parameters and return their sum. Demonstrate the usage of both methods in main().

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

class Math
{
public:
    int add(int a, int b) {
        return a + b;
    }
    int add(int a, int b, int c) {
        return a + b + c;
    }
};

int main() {
    Math math;
    int sum1 = math.add(20,8);
    cout << "Sum  of 20 and 8 is: " << sum1 << endl;
    int sum2 = math.add(69, 71, 96);
    cout << "Sum of 69,71,96 is: " << sum2 << endl;

    return 0;
}
```

```
Sum  of 20 and 8 is: 28
Sum of 69,71,96 is: 236
```

8. Implement a class Area with overloaded methods calculate(). The first method should take the radius of a circle as a parameter and return the area of the circle. The second method should take the length and width of a rectangle as parameters and return the area of the rectangle. Demonstrate the usage of both methods in main().

```cpp
#include<iostream>
using namespace std;
class Area {
public:
    double calculate(float radius) {
        return 3.14 * radius * radius;
    }
    double calculate(double length, double width) {
        return length * width;
    }
};
int main() {
    Area a;
    double circleArea = a.calculate(14.6);
    double rectangleArea = a.calculate(4.5, 9.5);
    cout << "Area of circle with radius 14.6: " << circleArea << endl;
    cout << "Area of rectangle with length 4.5 and width 9.5: " << rectangleArea << endl;
    return 0;
}
```

```
Area of circle with radius 14.6: 669.322
Area of rectangle with length 4.5 and width 9.5: 42.75
```

## Discussion

This lab introduced object-oriented programming fundamentals in C++. We created classes with private and public members, instantiated objects from these classes, and implemented different types of constructors (default, parameterized, and copy) for object initialization. We also used destructors to manage automatic cleanup when objects go out of scope, providing practical experience with core OOP concepts and memory management in C++.

## Conclusion

This lab gave us a solid understanding of how classes and objects work in C++. We learned how to create objects, set them up with constructors, and clean them up with destructors. Working through different programming examples helped us see how to keep data safe inside classes and solve problems using an object-oriented approach. Overall, it was a practical way to get comfortable with the basics of C++ programming.