TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

HIMALAYA COLLEGE OF ENGINEERING

CHYASAL, LALITPUR

Lab Report No: - 03

Title: - Class, Object, Constructor and Destructor

Submitted by: -

Name: - Diwas Pokhrel

Roll NO: -HCE081BEI014

Submitted To: -

Department of Electronics and Computer Engineering

Checked by: -

Date of submission: - 2081/03/01

OBJECTIVE

- To understand the concept of classes and objects in C++.
- To learn how to define constructors (default and parameterized) for object initialization.
- To study the role of destructors in object cleanup and their execution order.
- To implement class member functions for data manipulation.

THEORY

CLASS

A **class** in C++ is a **user-defined data type** that acts as a **blueprint** for creating objects. It can contain:

- **Data members** (variables to store data)
- **Member functions** (functions to perform operations on the data)
- A class groups related variables and functions together to model real-world entities.

Syntax:

```
class ClassName {
public:          // Access specifier
  // Data members
  // Member functions
};
```

OBJECT

An **object** is an **instance of a class**. When a class is defined, no memory is allocated. Memory is only allocated when an object is created.

- Objects can access the class members using the **dot operator** (.).

```
#include <iostream>
using namespace std;
// Define a class
class Person {
public:
string name;
int age;
// Member function
void display() {
cout << "Name: " << name << endl;
cout << "Age: " << age << endl;}
```

```
};
int main() {
// Create an object of class Person
Person p1;
p1.name = "Diwas";
p1.age = 20;
// Call member function
p1.display();
return 0;
}
```

Constructor

It is a special member function called automatically when an object is created.

Purpose:

- To initialize objects automatically when they are created
- To avoid manual initialization using a separate function
- To improve code readability, safety, and reusability

Types:

1. Default Constructor:
   - No parameters (initializes with default values).
2. Parameterized Constructor:
   - Takes arguments for initialization.
3. Copy Constructor :
   - Creates a new obejct as like a copy of existing object .

Example:

```
#include <iostream>

using namespace std;

class Person {

private:

string name;

int age;

public:
```

```
    // Default constructor
    Person() {
        name = "Unknown";
        age = 0;
    }.          // Parameterized constructor
    Person(string n, int a) {
        name = n;
        age = a;
    }       // Copy constructor
    Person(const Person &p) {
        name = p.name;
        age = p.age;
    }                                       // Display function
    void display() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }. };
int main() {                                // Object using default constructor
    Person p1;
    cout << "Default Constructor: ";
    p1.display();                           // Object using parameterized constructor
    Person p2("Diwas", 20);
    cout << "Parameterized Constructor: ";
    p2.display();
                                                    // Object using copy constructor
    Person p3 = p2;
cout << "Copy Constructor: ";
p3.display();
return 0;
}
```

Destructor

- A special member function called when an object is destroyed (e.g: program ends or delete is used .

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()

- Executes in reverse order of object creation.

Example :

```
class Demo {
int id;
public:
Demo(int i) {                    // Constructor
id = i;
cout << "Constructor: " << id << endl;
}
~Demo() {                        // Destructor
cout << "Destructor: " << id << endl;
}
};
int main() {
Demo d1(1), d2(2);          // Destructors called in reverse order (2 → 1)
return 0;}
```

- **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**

```
#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 "<<endl;
//cout<<"Brand name: "<<car1.brand<<endl;
//cout<<"Model: "<<car1.model<<endl;
//cout<<"Year: "<<car1.year<<endl;
//Cannot execute
cout<<"Displaying data members using member fucntion. "<<endl;
car1.displaydata();
return 0;
}
```

```
Output
Enter the details of the car :
Brand name: TATA
Model: EEe
Year: 2025
Displaying data members using member fucntion.
Brand name: TATA
Model: EEe
Year: 2025


=== Code Execution Successful ===
```

- **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**

```
1. #include<iostream>
2. #include<string>
3. using namespace std;
4. class Book {
5. string title;
6. string author;
7. int year;
8. public:
9. Book()
10. {
11. title=" ";
12. author=" ";
13. year=0;
14. }
15. Book(string a, string b, int c)
16. {
17. title=a;
18. author=b;
19. year=c;
20. }
21. void display()
22. {
23. cout<<"Title: "<<title<<endl;
```

```
24. cout<<"Author: "<<author<<endl;
25. cout<<"Year: "<<year<<endl;
26. }
27. };
28. int main()
29. {
30. Book b1;
31. cout<<"Display with default constructor: "<<endl;
32. b1.display();
33. Book b2("THE UPCOMING BOOK","Diwas pokhrel",2035);
34. cout<<"Display with parametetized constructor: "<<endl;
35. b2.display();
36. return 0;
37. }
```

Output

```
Display with default constructor:
Title:
Author:
Year: 0
Display with parametetized constructor:
Title: THE UPCOMING BOOK
Author: Diwas pokhrel
Year: 2035


=== Code Execution Successful ===
```

- **Create 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.**

```
1. #include <iostream>
2. #include <string>
3. using namespace std;
4. class Employee {
5. private:
6. string name;
7. int age;
8. public:
```

```
9.   Employee(string empName, int empAge) {
10. name = empName;
11. age = empAge;
12. }
13. Employee(const Employee& e) {
14. name = e.name;
15. age = e.age;
16. cout << "\n Called the copy constructor" << endl;
17. }
18. void display()  {
19. cout << "Employee Name: " << name << endl;
20. cout << "Employee Age : " << age << endl;
21. }
22. };
23. int main() {
24. Employee emp1("Diwas", 20);
25. cout << "Original Details of the Employee:\n";
26. emp1.display();
27. Employee emp2 = emp1;
28. cout << "\nCopied  Details of the Employee:\n";
29. emp2.display();
30. return 0;
31. }
```

Output

```
Original Details of the Employee:
Employee Name: Diwas
Employee Age : 20

 Called the copy constructor

Copied  Details of the Employee:
Employee Name: Diwas
Employee Age : 20


=== Code Execution Successful ===
```

- **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.**

```
1. #include <iostream>
2. #include <string>
3. using namespace std;
4. class Book {
5. private:
6. string title;
7. public:
8. Book(string t) {
9. title = t;
10. cout << "Book \"" << title << "\"created." << endl;
11. }
12. ~Book() {
13. cout << "Book \"" << title << "\" destroyed." << endl;
14. }
15. };
16. int main() {
17. cout << "CREATING :\n" << endl;
18. Book book1("THE UPCOMPING BOOK");
19. cout << "\n DESTROYING ()" << endl;
20. return 0;
21. }
```

Output

```
CREATING :

Book "THE UPCOMPING BOOK"created.

 DESTROYING ()
Book "THE UPCOMPING BOOK" destroyed.


=== Code Execution Successful ===
```

- **Define 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().**

```
1. #include <iostream>
2. using namespace std;
3. class Rectangle {
4. private:
5. int length;
6. int width;
7. public:
8. Rectangle(int l, int w) {
9. length = l;
10. width = w;
11. }
12. double calculateArea()  {
13. return length * width;
14. }
15. friend void doubleDimensions(Rectangle &rect);
16. void displayDimensions() {
17. cout << "Length: " << length << ", Width: " << width << endl;
18. }
19. };
20. void doubleDimensions(Rectangle &rect) {
21. rect.length *= 2;
22. rect.width *= 2;
23. }
24. int main() {
25. int l, w;
26. cout << "Enter the length of the rectangle: ";
27. cin >> l;
28. cout << "Enter the width of the rectangle: ";
29. cin >> w;
30. Rectangle myRect(l,w);
31. cout << "\n  The original dimensions is:" << endl;
32. myRect.displayDimensions();
```

```
33. cout << " Area of original length and width is : " << myRect.calculateArea() << endl;
34. doubleDimensions(myRect);
35. cout << "\n After doubling dimensions:" << endl;
36. myRect.displayDimensions();
37. cout << "The new area is: " << myRect.calculateArea() << endl;
38. return 0;
39. }
```

Output

```
Enter the length of the rectangle: 2
Enter the width of the rectangle: -1

  The original dimensions is:
Length: 2, Width: -1
 Area of original length and width is : -2

 After doubling dimensions:
Length: 4, Width: -2
The new area is: -8


=== Code Execution Successful ===
```

- **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 ().**

```
1. #include <iostream>
2. #include <string>
3. using namespace std;
4. class Student {
5. private:
6. string name;
7. int age;
8. public:
9. Student(string n, int a) {
10. name = n;
11. age = a;
12. }
13. Student() {
14. name = "";
15. age = 0;
16. }
17. void inputStudentDetails() {
18. cout << "Enter name: ";
19. cin.ignore();
20. getline(cin, name);
```

```
21. cout << "Enter age: ";
22. cin >> age;
23. }
24. void displayStudentDetails() {
25. cout << "Name: " << name << ", Age: " << age << endl;
    }
26. };
27. int main() {
28. const int SIZE = 3;
29. Student studentArray[SIZE];
30. cout << "Enter details for " << SIZE << " students:\n";
31. for (int i = 0; i < SIZE; ++i) {
32. cout << "\nStudent " << i + 1 << ":\n";
33. studentArray[i].inputStudentDetails();
34. }
35. cout << "\nStudent Details:\n";
36. for (int i = 0; i < SIZE; ++i) {
37. cout << "Student " << i + 1 << ": ";
38. studentArray[i].displayStudentDetails();
39. }
40. return 0;
41. }
```

Output

```
Enter details for 3 students:

Student 1:
Enter name: mukesh
Enter age: 19

Student 2:
Enter name: diwas
Enter age: 19

Student 3:
Enter name: atul
Enter age: 12

Student Details:
Student 1: Name: ukesh, Age: 19
Student 2: Name: diwas, Age: 19
Student 3: Name: atul, Age: 12


=== Code Execution Successful ===
```

- **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().**

```
1. #include <iostream>
2. using namespace std;
3. class Math {
4. public:
5. int add(int a, int b) {
6. return a + b;
7. }
8. int add(int a, int b, int c) {
9. return a + b + c;
10. }
11. };
12. int main() {
13. Math m;
14. int sum1 = m.add(19, 06);
15. cout << "Sum  of 19 and 06 is: " << sum1 << endl;
16. int sum2 = m.add(19, 06, 63);
17. cout << "Sum of 19,06,63 is: " << sum2 << endl;
18. return 0;
19. }
```

Output

```
Sum  of 19 and 06 is: 25
Sum of 19,06,63 is: 88


=== Code Execution Successful ===
```

- **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().**

```
1. #include<iostream>
2. using namespace std;
3. class Area {
4. public:
5. double calculate(double radius) {
6. return 3.14 * radius * radius;
7. }
8. double calculate(double length, double width) {
9. return length * width;
10. }
11. };
12. int main() {
13. Area a;
14. double circleArea = a.calculate(7.7);
15. double rectangleArea = a.calculate(3.4, 5.5);
16. cout << "Area of circle with radius 7.7: " << circleArea << endl;
17. cout << "Area of rectangle with length 3.4 and width 5.5: " << rectangleArea << endl;
18. return 0;}
```

Output

```
Area of circle with radius 7.7: 186.171
Area of rectangle with length 3.4 and width 5.5: 18.7


=== Code Execution Successful ===
```

Discussion:

In this lab, we explored the foundational concept of object-oriented programming in C++ through the use of classes and objects. We learned how to define classes with private and public members, and how objects serve as instances of those classes. We implemented various constructors—default, parameterized, and copy—to initialize objects in different ways. The use of destructors was also demonstrated, showing how they automatically handle cleanup at the end of an object's lifecycle.

Conclusion:

This lab successfully demonstrated the core principles of classes and objects in C++. It helped in understanding how constructors initialize objects, how destructors manage cleanup, and how member functions operate on object data. By completing various class-based programs, we strengthened our grasp on encapsulation, data abstraction, and the overall object-oriented approach to problem-solving in C++.