Himalaya College of Engineering

Advanced C++ Programming Lab Report

Lab 3: Class, Object, Constructor and Destructor in C++

Prepared By : Nawnit Paudel (HCE081BEI024)

Subject : Object-Oriented Programming (OOP)

Program : Bachelor of Electronics, Communication and Information Engineering

Institution : Himalaya College of Engineering

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 objectoriented 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;

    }}; int main() {
Demo obj;
obj.setValues();
obj.display();
     // obj.a = 1;  //Error: private member
// obj.b = 2;  // Error: protected member
```

obj.c = 3; // OK: public member

return 0;

}

Explanation:

- `a` is private → Only accessible inside class methods.
- `b` is protected → Also accessible inside the class and in derived classes.
- `c` is public → Can be accessed directly from outside the class (like in `main()`)

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

LAB ASSIGNMENTS:

Q1.) 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 {

private:

    string brand;

    string model;

    int year;
```

```
public:
    void setBrand(string b) {
        brand = b;
    }

    void setModel(string m) {
        model = m;
    }

    void setYear(int y) {
        year = y;
    }

    string getBrand() {
        return brand;
    }

    string getModel() {
        return model;
    }

    int getYear() {
```

```
        return year;
    }
};

int main() {
    Car myCar;
    myCar.setBrand("Toyota");
    myCar.setModel("Corolla");
    myCar.setYear(2020);

    cout << "Brand: " << myCar.getBrand()
<< endl;
    cout << "Model: " << myCar.getModel()
<< endl;
    cout << "Year: " << myCar.getYear() <<
endl;

    return 0;
}
```

Q2.) 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 {
private:
    string title;
    string author;
    int year;

public:
    Book() {
        title = "The Extreme Curses";
        author = "NeverKnown";
        year = 1;
    }

    Book(string t, string a, int y) {
        title = t;
        author = a;
        year = y;
    }

    void displayDetails() {
        cout << "Title: " << title << endl;
        cout << "Author: " << author << endl;
        cout << "Year: " << year << endl;
    }
};

int main() {
    Book defaultBook;
    Book customBook("1984", "George Orwell", 1949);
```

```
    cout << "Default Book:" << endl;
    defaultBook.displayDetails();

    cout << "\nCustom Book:" << endl;
    customBook.displayDetails();

    return 0;
```

}**Q3.) 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.**

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

class Employee {
private:
    string name;
    int age;

public:
    Employee(string n, int a) {
        name = n;
        age = a;
    }

    Employee(Employee& e) {
        name = e.name;
        age = e.age;
    }

    void display() {
```

```
        cout << "Name: " << name << endl;
        cout << "Age: " << age << endl;
    }
};

int main() {
    Employee e1("Nawnit", 19);
    Employee e2 = e1;

    cout << "Original Employee:" << endl;
    e1.display();

    cout << "\nCopied Employee:" << endl;
    e2.display();

    return 0;
```

}**Q4.) 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.**

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

class Book {
private:
    string title;

public:
    Book(string t) {
        title = t;
        cout << "Book: " << title << endl;
```

```
    }

    ~Book() {
        cout << "Book: " << title << endl;
    }
};

int main() {
    Book b1("The Tale Of Sanit");

    {
        Book b2("2024");
        Book b3("What’s up dude?");
    }

    return 0;
```

}**Q5.) 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().**

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

class Rectangle {
private:
    double length;
    double width;

public:
    Rectangle(double l, double w) {
        length = l;
```

```
        width = w;
    }

    double Area() {
        return length * width;
    }

    void Dimensions() {
        length *= 2;
        width *= 2;
    }

    void display() {
        cout << "Length: " << length << ", Width: " << width << endl;
    }
};

void Dimensions(Rectangle& rect) {
    rect.Dimensions();
}

int main() {
    Rectangle r1(4.0, 5.0);

    cout << "Original dimensions:" << endl;
    r1.display();
    cout << "Area: " << r1.Area() << endl;

    Dimensions(r1);

    cout << "\nAfter doubling dimensions:" << endl;
```

```
    r1.display();
    cout << "Area: " << r1.Area() << endl;

    return 0;
```

}**Q6.) 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;
    }

    void studentDetails() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};

int main() {
    Student studentArray[3] = {
        Student("Nawnit", 19),
        Student("Sadikshya", 18),
```

```
        Student("Sanit", 0)
    };

    for (int i = 0; i < 3; i++) {
        studentArray[i].studentDetails();
    }

    return 0;
```

}**Q7.) 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().**

```
#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 m;

    cout << "Sum of 5 and 10: " << m.add(5, 10) << endl;
    cout << "Sum of 2, 4 and 6: " << m.add(2, 4, 6) << endl;
```

```
    return 0;
```

}**Q8.) 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().**

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

class Area {
public:
    double
calculate(double
radius) {
        return 3.14 *
radius * radius;
    }

    double
calculate(double
length, double width)
{
        return length *
width;
    }
};

int main() {
    Area a;
    cout << "Area of
circle with radius 3: "
<< a.calculate(3.0) <<
endl;
```

```
    cout << "Area of
rectangle 4 x 5: " <<
a.calculate(4.0, 5.0) <<
endl;


    return 0;
}
```

DISCUSSION:

A class is like a plan or recipe for making things called objects. Each object is made from the class and has its own information. A constructor is a special tool that gets the object ready when it is made. A destructor is a tool that cleans up when the object is no longer needed. These ideas help us write clear and easy-to-understand programs.

CONCLUSION:

A class is like a plan or recipe for making things called objects. Each object is made from the class and has its own information. A constructor is a special tool that gets the object ready when it is made. A destructor is a tool that cleans up when the object is no longer needed. These ideas help us write clear and easy-to-understand programs.