Objectives:

- To familiarize with the concept of classes and objects in C++.
- To implement constructors for automatic object initialization.
- To use destructors for freeing resources and understanding object lifecycle.
- To promote modular and reusable code by employing object-oriented principles.

Tools and Libraries Used:

- Programming Language: C++
- IDE: G++
- Libraries: include <iostream>, include <string>

Theory:

In C++, classes serve as custom data types that encapsulate both data (attributes) and functions (behaviors). A constructor is a special method that runs automatically upon object initialization, used to set up its initial state. The destructor is called when the object goes out of scope or is destroyed, freeing resources or performing cleanup tasks. This combination forms the foundation of object-oriented programming, enabling code reuse, data abstraction, and ease of maintenance.

Object

In OOP, a problem is divided into a set of such entities, each retaining its own data and functionality.

- Objects are the main elements that come into action during a program's execution.
- Communication between objects happens through messages.
- An object's methods can directly access its own data members.

Class

A class acts as a template or a blueprint for creating numerous objects with similar properties and methods. Each object instantiated from a class maintains its own separate data.

- Once a class is defined, you can create multiple instances of it.
- A class typically controls access to its members through public, protected, or private visibility.
- This forms the basis for data encapsulation and information hiding.

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;
}
```

Constructors

- A constructor is a special member function in C++.
- It is called automatically when a new object is created.
- The main role of a constructor is to initialize its data members.

Purpose:

- Initializes the object's attributes when it's created.
- Allows you to set default or custom values for its members.
- Helps streamline object instantiation when multiple copies are needed.

Characteristics:

- The constructor's name is the same as its class.
- It's automatically called upon object instantiation.
- If you do not provide a custom constructor, the compiler generates a default one.
- The constructor doesn't have any return type.

Syntax:

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

Types of Constructors

a. Default constructor

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

b. Parameterized constructor

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

c. Copy constructor

```
class ClassName {
public:
    ClassName(const ClassName &obj);
};
```

Destructor

- The destructor is a special method in C++.
- It is automatically called when an object is destroyed.
- It performs clean-up operations and frees resources, if needed.

Purpose:

- Allows proper release of resources (like closing files or freeing memory).
- Helps avoid memory leaks in programs employing dynamic memory.

Characteristics:

- The destructor's name is the class's name with a tilde (~).
- It does not take any parameters and returns nothing.
- Each class can have at most one destructor.
- It's called automatically when the object goes out of scope or is destroyed.

Syntax:

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

Example:

```
#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(); // Method is called
    return 0; // Destructor is called automatically
}
```

Lab Questions:

Q no 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.

Code:

```
#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: "<<endl;
    car1.setdata();
    //cout<<"Displaying datamembers directly"<<endl;
    //cout<<"Brand name: "<<car1.brand<<endl;
    //cout<<"Model: "<<car1.model<<endl;
    //cout<<"Year: "<<car1.year<<endl;
        // This is the wrong way and won't execute with this code.
    cout<<"Displaying datamembers using member fucntion. "<<endl;
    car1.displaydata();
    return 0;
}
```

Output:

```
Enter the details:
Brand name: HCOE
Model: BEI
Year: 2081
Displaying datamembers using member fucntion.
Brand name: HCOE
Model: BEI
Year: 2081
```

Q no 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.

Code:

```
#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("Muna_Madan","Laxmi_Parsad_Devkota",1999);
       cout<<"Display with parametetized constructor: "<<endl;
       b2.display();
       return 0;
   }
```

Output:

```
Display with default constructor:
Title:
Author:
Year: 0
Display with parametetized constructor:
Title: Muna_Madan
Author: Laxmi_Parsad_Devkota
Year: 1999
```

Q no 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.

Code:

```
#include<iostream>
#include<string>
using namespace std;
class Employee {
    string name;
    int age;
public:
    void setdata(string a, int b) {
        name = a;
           age = b;
       }
       Employee(const Employee &obj) {
           name = obj.name;
           age = obj.age;
       }
       void display() {
           cout << "Name: " << name << endl;
           cout << "Age: " << age << endl;
       }
       Employee() {}
   };
   int main() {
       Employee e1;
       e1.setdata("Ramesh", 50);
       cout << "Original data: " << endl;
       e1.display();
       Employee e2 = e1;
       cout << "Copied data: " << endl;
       e2.display();
       return 0;
   }
```

Output:

```
Original data:
Name: Ramesh
Age: 50
Copied data:
Name: Ramesh
Age: 50
```

Q no 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.

Code:

```
#include<iostream>
#include<string>
using namespace std;
class Book {
private:
    string title;
public:
    Book(string t) {
        title = t;
           cout << "Book \"" << title << "\" is created." << endl;
       }
       ~Book() {
           cout << "Book \"" << title << "\" is destroyed." << endl;
       }
   };
   int main()
       {
       Book b1("C++ Fundamentals");
       return 0;
   }
```

Output:

```
Book "C++ Fundamentals" is created.
Book "C++ Fundamentals" is destroyed.
```

Q no 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().

Code:

```
#include<iostream>
using namespace std;
class Rectangle {
    int length;
    int width;
    public:
    void setdata()
    {
        cout<<"Length: ";
           cin>>length;
           cout<<"Width: ";
           cin>>width;
       }
       void displaydata()
       {
           cout<<"Length: "<<length<<endl;
           cout<<"Width: "<<width<<endl;
       }
       Rectangle()
       {
           length=0;
           width=0;
       }
       void calculateArea()
       {
           int area=length*width;
           cout<<"Area is: "<<area<<endl;
       }
       void doubleDimensions(Rectangle &rect)
       {
           rect.length=rect.length*2;
           rect.width=rect.width*2;
       }
   };
   int main()
   {
       Rectangle r1;
       cout<<"Enter input: "<<endl;
       r1.setdata();
       cout<<"Area calculation: "<<endl;
       r1.calculateArea();
       cout<<"Double dimensions: "<<endl;
       r1.doubleDimensions(r1);
       r1.displaydata();
```

```
45.       cout<<"Area calculation: "<<endl;
46.       r1.calculateArea();
47.       return 0;
48.   }
```

Output:

```
Enter input:
Length: 5
Width: 2
Area calculation:
Area is: 10
Double dimensions:
Length: 10
Width: 4
Area calculation:
Area is: 40
```

Q no 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().

Code:

```
#include<iostream>
#include<string>
using namespace std;
class Student {
private:
    string name;
    int age;
public:
    Student() {
            name = "";
            age = 0;
        }
        void displayStudentDetails()
        {
            cout << "Name: " << name << ", Age: " << age <<
endl;
        }
        void setDetails() {
            cout << "Enter name: ";
            getline(cin, name);
            cout << "Enter age: ";
            cin >> age;
            cin.ignore();
        }
   };
   int main() {
        Student studentArray[3];
        cout << "Enter details of 3 students:" << endl;
        for (int i = 0; i < 3; i++) {
            cout << "Student " << i + 1 << ":" << endl;
            studentArray[i].setDetails();
        }
        cout << "Displaying student details:" << endl;
        for (int i = 0; i < 3; i++) {
            cout << "Student " << i + 1 << ": ";
            studentArray[i].displayStudentDetails();
        }
        return 0;
   }
```

Output:

```
Enter details of 3 students:
Student 1:
Enter name: Mukesh
Enter age: 20
Student 2:
Enter name: Diwas
Enter age: 25
Student 3:
Enter name: Gaurab
Enter age: 15
Displaying student details:
Student 1: Name: Mukesh, Age: 20
Student 2: Name: Diwas, Age: 25
Student 3: Name: Gaurab, Age: 15
```

Q no 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().

Code:

```
#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;
       int sum1 = m.add(5, 10);
       int sum2 = m.add(3, 6, 9);
       cout << "Sum of 5 and 10: " << sum1 << endl;
       cout << "Sum of 3, 6 and 9: " << sum2 << endl;
       return 0;
   }
```

Output:

```
Sum of 5 and 10: 15
Sum of 3, 6 and 9: 18
```

Q no 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().

Code:

```
#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;
       double circleArea = a.calculate(5.0);
       double rectangleArea = a.calculate(4.0, 6.0);
       cout << "Area of circle with radius 5: " << circleArea
<< endl;
       cout << "Area of rectangle with length 4 and width 6: "
<< rectangleArea << endl;
       return 0;
   }
```

Output:

```
Area of circle with radius 5: 78.5
Area of rectangle with length 4 and width 6: 24
```

Conclusion:

This lab provided a comprehensive understanding of the fundamental concepts of object-oriented programming in C++, particularly focusing on classes, objects, constructors, destructors, and method overloading. Through hands-on implementation, we explored how constructors streamline object initialization, how destructors help manage resource cleanup, and how encapsulation is achieved using access specifiers. Each program reinforced the principles of modularity and reusability, enabling efficient code design. Overall, the lab strengthened our grasp of key OOP principles and prepared us for more advanced programming techniques in C++.