TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

HIMALAYA COLLEGE OF ENGINEERING

CHYASAL, LALITPUR

Lab Report No: - 03
Title: - Objects and Classes

Submitted by: -
Name: - Bikram Panthi
Roll NO: - 13

Submitted To: -
Department Of
Checked by: -

Date of submission: -

Objectives:

- Understand the concept of classes and objects in C++.
- Learn how to define member functions inside and outside a class.
- Demonstrate access specifiers (public, private, protected) for data encapsulation.

Theory:

Class:

A **class** in C++ is a fundamental concept in object-oriented programming. It acts as a blueprint or a template for creating objects. When we think of a class, we can imagine it like a design or plan — it doesn't represent anything by itself but describes what an object of that class will contain and what actions it can perform.

In a class, we define variables (called data members) that store information, and functions (called member functions or methods) that operate on that information. These members define the properties and behaviors that the objects created from the class will have.

For example, if you are designing a program to manage information about books, you might create a class called Book with data members like title, author, and price. The class might also include functions to display details of the book or apply discounts.

Classes help organize code in a better way by grouping related data and functions together. It also helps in reusability — once a class is created, we can create many objects from it without rewriting the structure again. A class itself does not hold values, but it tells what kind of values and functions each object of that class will have.

Using classes brings several advantages like data hiding, modularity, abstraction, and encapsulation, making programs more structured and easier to manage.

Basic syntax:

```
class ClassName {
private:
// data members
public:
// member functions
};
```

Object:

An **object** is an actual instance created from a class. While a class is just a blueprint, an object is the real thing — a variable or entity that holds actual data and can perform real actions using the functions defined in the class.

Objects have their own separate copy of the data members defined in the class, meaning each object stores its own information. For example, if we have a class Book, then book1 and book2 can be two different objects of that class. Both will have the same structure (like having title, author, price), but the actual values can be different for each.

When we create an object, we are allocating memory and giving life to the structure defined by the class. Objects can then interact with each other or with the user. They are the actual working units in a program.

Objects are used to access the functions and data of a class. Without creating objects, the class remains just a design. You can create as many objects as you need from one class, and each one can behave independently.

In real-world terms, if a class is like a vehicle blueprint, then objects are the actual vehicles built from it — each one can have a different color, speed, or model, but all are built from the same design.

Basic syntax:

```
ClassName objectName; // object declaration
objectName.memberFunction(); // function call using object
```

Access Specifiers:

Access specifiers are keywords in C++ that define how the members (variables and functions) of a class can be accessed from different parts of the program. They help in controlling access to the data inside a class, which is very important for **security, encapsulation**, and **data hiding**.

There are **three main access specifiers** in C++:

1. Public

Members declared as public can be accessed from **anywhere** in the program — inside or outside the class. If we want some data or functions to be openly available to all parts of the code, we make them public. Usually, most functions (methods) that perform actions are made public so that users can call them.

2. Private

Members declared as private can be accessed **only from within the same class**. They are hidden from the outside world. This is useful when we want to protect the data from being directly modified or accessed. For example, sensitive data like passwords, or internal variables, are usually made private. Outside code can't access them directly — it must go through public functions to get or set the data safely.

3. Protected

Members declared as protected are **similar to private**, but with one extra feature: they can also be accessed by **derived (child) classes** during inheritance. This is useful when we want to restrict access but still allow subclasses to use or modify the data.

Access specifiers help us manage how the data and behavior of a class are exposed.

- public: open access
- private: hidden from outside
- protected: hidden, but visible in inherited classes

Using the right access specifier helps us write safe, organized, and secure object-oriented programs.

Q1. Define a class Car with private members brand, model, 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>
using namespace std;

class Car {
private:
    string brand;
    string model;
    int year;

public:
    // Setter functions
    void setBrand(string b) {
        brand = b;
    }

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

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

    // Getter functions
    string getBrand() {
        return brand;
```

```
    }

    string getModel() {
        return model;
    }

    int getYear() {
        return year;
    }

    // Display car details
    void displayCarInfo() {
        cout << "Car Brand: " << brand
            << endl;
        cout << "Car Model: " << model
            << endl;
        cout << "Car Year: " << year <<
            endl;
    }
};

int main() {
    Car car1;

    // Setting values using setter
        functions
```

```
    car1.setBrand("Toyota");
    car1.setModel("Corolla");
    car1.setYear(2020);

    // Displaying values using member
        function
    car1.displayCarInfo();

    return 0;
}
```

Output:

```
Car Brand: Toyota
Car Model: Corolla
Car Year: 2020


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

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.

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

class Book {
private:
    string title;
    string author;
    int year;

public:
    // Default constructor
    Book() {
        title = "Unknown Title";
        author = "Unknown Author";
        year = 0;
    }

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

    // Method to display book details
    void displayDetails() {
```

```
        cout << "Title: " << title <<
            endl;
        cout << "Author: " << author <<
            endl;
        cout << "Year: " << year <<
            endl;
    }
};

int main() {
    // Using default constructor
    Book book1;
    cout << "Book 1 (Default
        Constructor):" << endl;
    book1.displayDetails();

    cout << endl;

    // Using parameterized constructor
    Book book2("The Alchemist", "Paulo
        Coelho", 1988);

    cout << "Book 2 (Parameterized
        Constructor):" << endl;
    book2.displayDetails();

    return 0;
}
```

Output:

```
Book 1 (Default Constructor):
Title: Unknown Title
Author: Unknown Author
Year: 0

Book 2 (Parameterized Constructor):
Title: The Alchemist
Author: Paulo Coelho
Year: 1988


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

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**.**

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

class Employee {
private:
    string name;
    int age;

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

    // Copy constructor
    Employee(const Employee &emp) {
        name = emp.name;
        age = emp.age;
    }

    // Method to display employee
        details
    void displayDetails() {
        cout << "Name: " << name <<
            endl;
```

```
25          cout << "Age: " << age << endl;
26      }
27  };
28
29  int main() {
30      // Create original employee object
31      Employee emp1("Bikram Panthi", 21);
32
33      // Create copy using copy
            constructor
34      Employee emp2 = emp1;
35
36      cout << "Original Employee Details
            :" << endl;
37      emp1.displayDetails();
38
39      cout << "\nCopied Employee Details
            :" << endl;
40      emp2.displayDetails();
41
42      return 0;
43  }
```

Output:

```
Original Employee Details:
Name: Bikram Panthi
Age: 21

Copied Employee Details:
Name: Bikram Panthi
Age: 21


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

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.

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

class Book {
private:
    string title;

public:
    // Constructor
    Book(string t) {
        title = t;
        cout << "Book \"" << title <<
            "\" is created." << endl;
    }

    // Destructor
    ~Book() {
        cout << "Book \"" << title <<
            "\" is destroyed." << endl;
    }

    // Display function
    void displayTitle() {
        cout << "Book Title: " << title
            << endl;
    }
```

```
};

int main() {
    // Creating objects inside a block to see destructor call
    {
        Book book1("C++ Programming");
        Book book2("Object Oriented Concepts");

        book1.displayTitle();
        book2.displayTitle();
    } // Destructor is called here automatically when the block ends

    cout << "End of main function." << endl;
    return 0;
}
```

Output:

```
Book "C++ Programming" is created.
Book "Object Oriented Concepts" is created.
Book Title: C++ Programming
Book Title: Object Oriented Concepts
Book "Object Oriented Concepts" is destroyed.
Book "C++ Programming" is destroyed.
End of main function.


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

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().

```
1  #include <iostream>
2  using namespace std;
3
4  class Rectangle {
5  private:
6      float length;
7      float width;
8
9  public:
10     // Constructor to initialize
           members
11     Rectangle(float l, float w) {
12         length = l;
13         width = w;
14     }
15
16     // Function to calculate area
17     float calculateArea() {
18         return length * width;
19     }
20
21     // Function to double the
           dimensions of a rectangle
22     void doubleDimensions(Rectangle
           rect) {
23         length = rect.length * 2;
```

```
24          width = rect.width * 2;
25      }
26
27      // Function to display length and
            width
28      void display() {
29          cout << "Length: " << length <<
                ", Width: " << width <<
                endl;
30      }
31  };
32
33  int main() {
34      // Creating a rectangle object with
            initial dimensions
35      Rectangle rect1(4.5, 3.0);
36      cout << "Original Rectangle:" <<
            endl;
37      rect1.display();
38
39      // Calculating and displaying area
40      float area = rect1.calculateArea();
41      cout << "Area of Rectangle: " <<
            area << endl;
42
43      // Creating another rectangle to
            double from
44      Rectangle rect2(2.0, 1.5);
45      rect1.doubleDimensions(rect2);
46
47      cout << "\nAfter Doubling
            Dimensions using rect2:" <<
            endl;
48      rect1.display();
49
50      return 0;
51  }
```

Output:

```
Original Rectangle:
Length: 4.5, Width: 3
Area of Rectangle: 13.5

After Doubling Dimensions using rect2:
Length: 4, Width: 3


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

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().

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

class Student {
private:
    string name;
    int age;

public:
    // Constructor to initialize name and age
    Student(string n, int a) {
        name = n;
        age = a;
    }

    // Default constructor (needed for array initialization)
    Student() {
        name = "";
        age = 0;
    }

    // Method to set student details (for array population)
    void setDetails(string n, int a) {
```

```
24          name = n;
25          age = a;
26      }
27
28      // Method to display student
            details
29      void displayStudentDetails() {
30          cout << "Name: " << name << ",
                Age: " << age << endl;
31      }
32  };
33
34  int main() {
35      // Array of 3 Student objects
36      Student studentArray[3];
37
38      // Populating the array with
            student details
39      studentArray[0].setDetails("Ram",
            18);
40      studentArray[1].setDetails("Sita",
            19);
41      studentArray[2].setDetails("Hari",
            20);
```

```
42
43      // Displaying details
44      cout << "Student Details:" << endl;
45      for (int i = 0; i < 3; i++) {
46          studentArray[i]
                .displayStudentDetails();
47      }
48
49      return 0;
50  }
```

Output:

```
Student Details:
Name: Ram, Age: 18
Name: Sita, Age: 19
Name: Hari, Age: 20


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

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().

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

class Math {
public:
    // Method to add two integers
    int add(int a, int b) {
        return a + b;
    }

    // Overloaded method to add three integers
    int add(int a, int b, int c) {
        return a + b + c;
    }
};

int main() {
    Math m;

    // Calling the method with 2 parameters
    int sum1 = m.add(10, 20);
    cout << "Sum of 10 and 20 is: " << sum1 << endl;

```

```
24      // Calling the overloaded method
            with 3 parameters
25      int sum2 = m.add(5, 15, 25);
26      cout << "Sum of 5, 15 and 25 is: "
            << sum2 << endl;
27
28      return 0;
29  }
```

Output:

```
Sum of 10 and 20 is: 30
Sum of 5, 15 and 25 is: 45


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

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().

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

class Area {
public:
    // Method to calculate area of a circle
    float calculate(float radius) {
        return 3.14 * radius * radius;
    }

    // Overloaded method to calculate area of a rectangle
    float calculate(float length, float width) {
        return length * width;
    }
};

int main() {
    Area a;

    // Area of circle
    float circleArea = a.calculate(5.0); // radius = 5
```

```
22      cout << "Area of Circle (radius 5):
            " << circleArea << endl;
23
24      // Area of rectangle
25      float rectangleArea = a.calculate(4
            .0, 6.0); // length = 4, width
            = 6
26      cout << "Area of Rectangle (4 x 6):
            " << rectangleArea << endl;
27
28      return 0;
29  }
```

Output:

```
Area of Circle (radius 5): 78.5
Area of Rectangle (4 x 6): 24


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

Discussion:

In this lab, we explored how classes and objects work in C++. We focused on method overloading, where functions with the same name perform different tasks depending on the number or type of parameters. This helps us write cleaner and more organized code. For example, we used the same calculate() function to find the area of both a circle and a rectangle. This shows how C++ supports multiple functions with the same name inside a class, as long as their parameter list is different.

Conclusion:

This lab helped us understand the basics of object-oriented programming like creating classes, objects, and using function overloading. It made clear how we can use the same function name in different ways, depending on what we pass to it. This makes our code more readable and flexible.