TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

HIMALAYA COLLEGE OF ENGINEERING

CHYASAL, LALITPUR

Lab Report No: -04

Title: - Inheritance and Its Types

Submitted by: -	**Submitted To: -**
Name: - Diwas Pokhrel	**Department of Electronics and Computer Engineering**
Roll NO: - HCE081BEI014	**Checked by: -**

Date of submission: - 2082/03/8

OBJECTIVE:

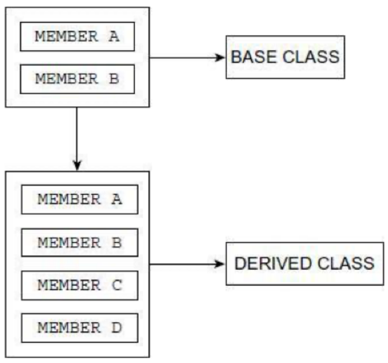
• To understand the concept of inheritance in object-oriented programming using C++.
• To explore and implement different types of inheritance .
• To learn how inheritance promotes code reuse and reduces redundancy in programming.
• To understand how data and functions can be inherited from one class to another.

TOOLS AND LIBRARIES USED:

• Programming Language C++
• IDE: Clang
• Libraries : include<iostream>,include<string>.

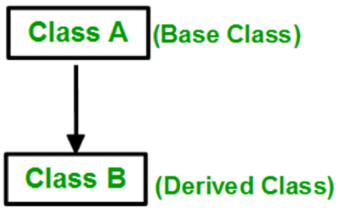
THEORY

Inheritance is one of the most important features of object-oriented programming (OOP), and it plays a vital role in creating efficient, reusable, and structured code . It allows derived class to acquire the properties (data members) and behaviors (member functions)of the base class. The programmer can define new member variables and functions in the derived class. The base class remains unchanged. The object of the derived class can access members of the base as well as derived classes. On the other hand, the object of the base class cannot access members of the derived classes. The base class does not know about their subclasses.
In C++, inheritance is implemented using the ":" symbol, followed by a visibility mode like "public", "protected", or "private". It helps in extending existing functionality without rewriting the same code again



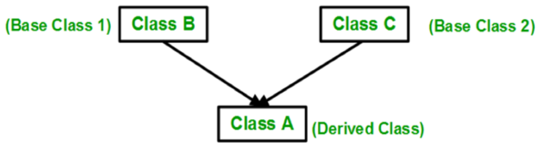
Types of Inheritance in C++:

1. Single Inheritance:
A single derived class inherits from a single base class. This is the simplest form and is useful when one class extends another class's functionality



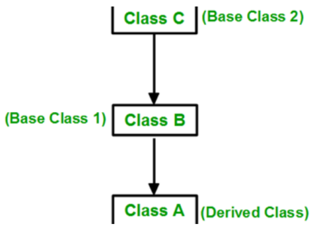
2. Multiple Inheritance:

A derived class inherits from more than one base class. It helps in combining features from multiple sources but can also lead to complexity such as the diamond problem

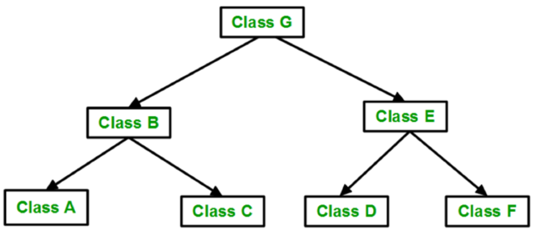


3. Multilevel Inheritance:

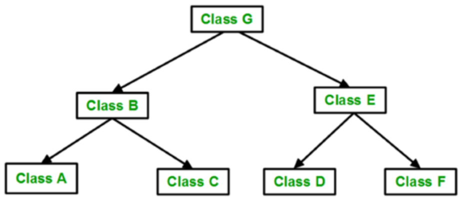
A class is derived from a class which is already a derived class. It creates a chain-like relationship (e.g., Grandparent → Parent → Child).



4. Hierarchical Inheritance:
Multiple derived classes are created from a single base class. This is commonly used when different classes share common behavior but have their own specific features



5. Hybrid Inheritance:
A combination of two or more types of inheritance. It reflects complex real-world relationships and often involves careful class design to avoid ambiguity.



Importance of Inheritance:

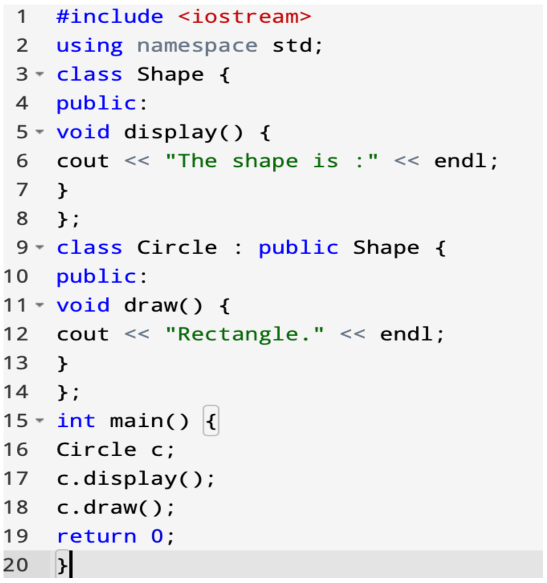
• Helps in code reusability i.e no need to write the same code again.

• Makes the code more manageable and readableby logically grouping related classes.

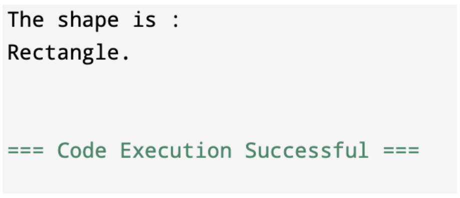
• Encourages extensibility – new functionality can be added to existing code without modifying it.

Exercise 1: **Single Inheritance**

- Create a base class Shape with a method display().
- Create a derived class Circle that inherits from Shape and has an additional method
- draw().
- Implement a main() function to demonstrate the usage of these classes.



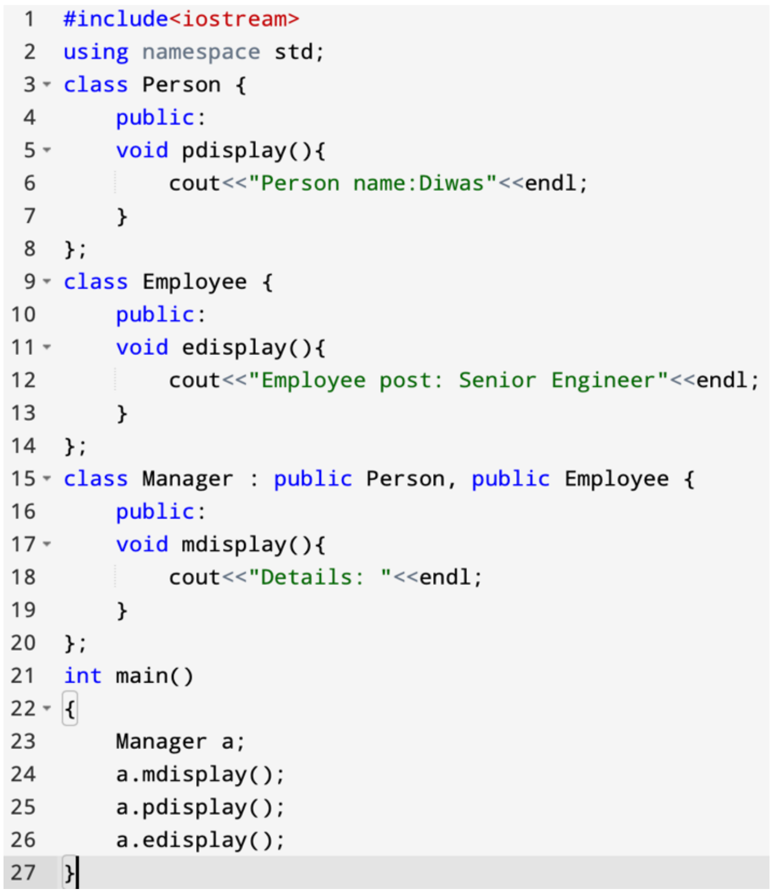
```
#include <iostream>
using namespace std;
class Shape {
public:
void display() {
cout << "The shape is :" << endl;
}
};
class Circle : public Shape {
public:
void draw() {
cout << "Rectangle." << endl;
}
};
int main() {
Circle c;
c.display();
c.draw();
return 0;
}
```



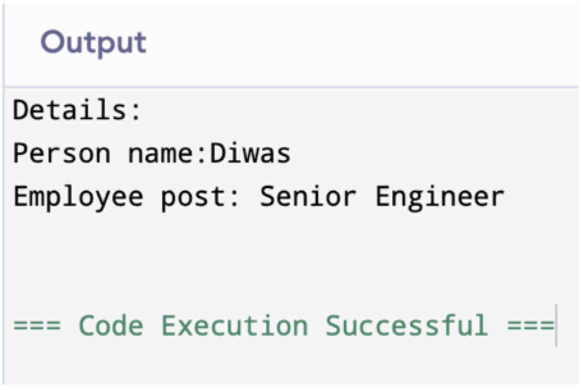
```
The shape is :
Rectangle.


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

Exercise 2: **Multiple Inheritance**

- Create two base classes Person and Employee with appropriate methods.
- Create a derived class Manager that inherits from both Person and Employee.
- Implement a main() function to demonstrate the usage of these classes.



```cpp
#include<iostream>
using namespace std;
class Person {
    public:
    void pdisplay(){
        cout<<"Person name:Diwas"<<endl;
    }
};
class Employee {
    public:
    void edisplay(){
        cout<<"Employee post: Senior Engineer"<<endl;
    }
};
class Manager : public Person, public Employee {
    public:
    void mdisplay(){
        cout<<"Details: "<<endl;
    }
};
int main()
{
    Manager a;
    a.mdisplay();
    a.pdisplay();
    a.edisplay();
}
```

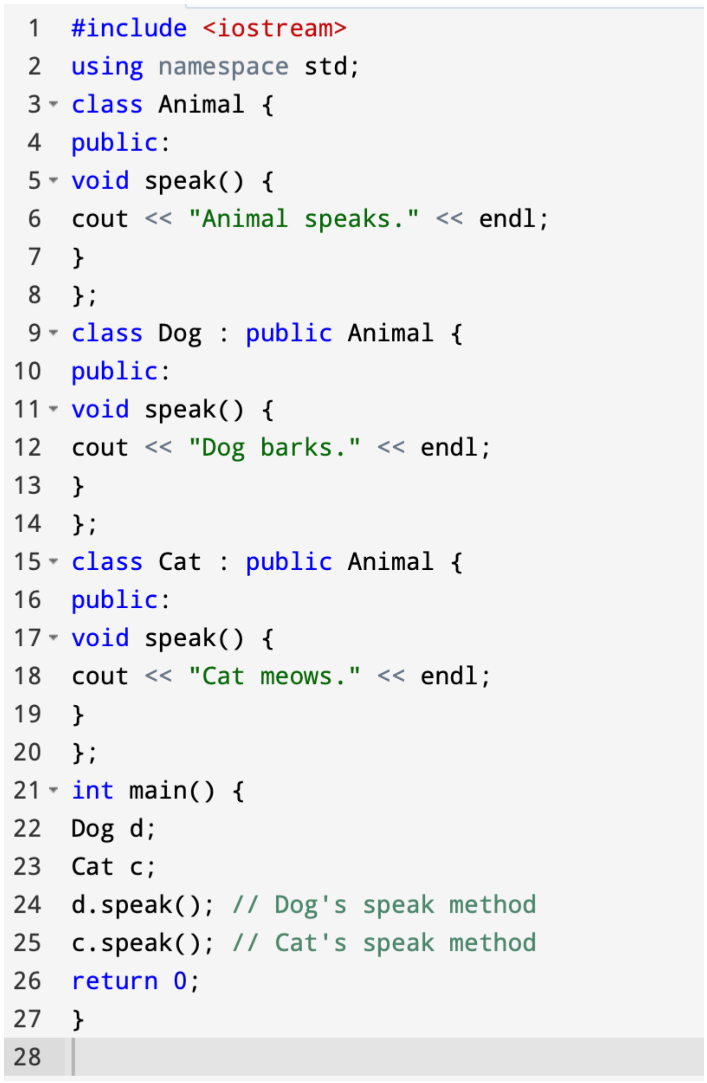


```
Details:
Person name:Diwas
Employee post: Senior Engineer


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

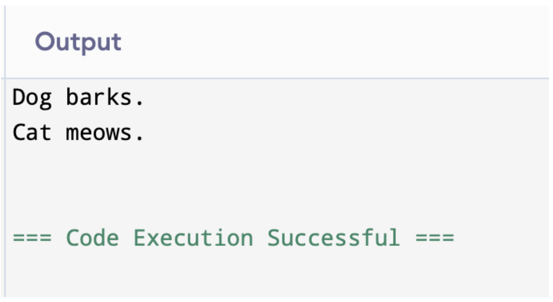
Exercise 3: **Hierarchical Inheritance**

- Create a base class Animal with a method speak().
- Create two derived classes Dog and Cat that inherit from Animal and have their own
- speak() methods.

Implement a main() function to demonstrate the usage of these classes.



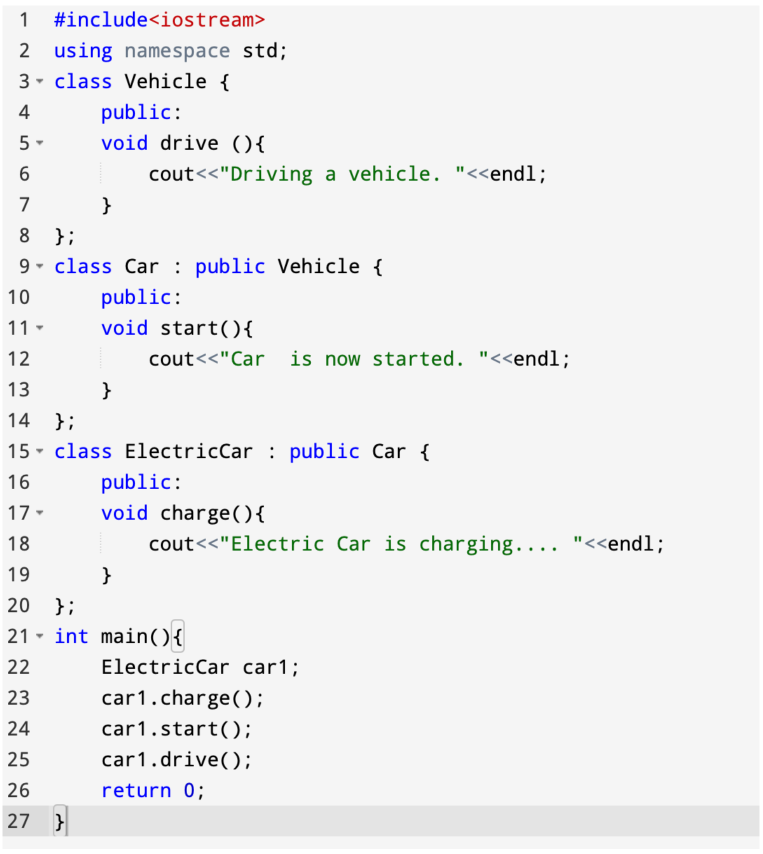
```cpp
#include <iostream>
using namespace std;
class Animal {
public:
void speak() {
cout << "Animal speaks." << endl;
}
};
class Dog : public Animal {
public:
void speak() {
cout << "Dog barks." << endl;
}
};
class Cat : public Animal {
public:
void speak() {
cout << "Cat meows." << endl;
}
};
int main() {
Dog d;
Cat c;
d.speak(); // Dog's speak method
c.speak(); // Cat's speak method
return 0;
}
```



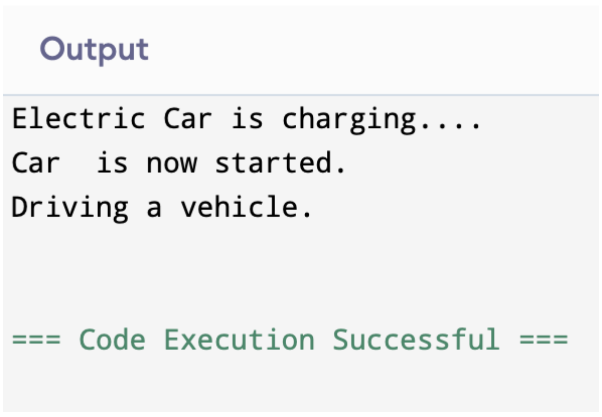
```
Dog barks.
Cat meows.


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

Exercise 4: **Multilevel Inheritance**

- Create a base class Vehicle with a method drive().
- Create a derived class Car that inherits from Vehicle and has an additional method
- start().
- Create another derived class ElectricCar that inherits from Car and adds its own
- method charge().
- Implement a main() function to demonstrate the usage of these classes.



```cpp
#include<iostream>
using namespace std;
class Vehicle {
    public:
    void drive (){
        cout<<"Driving a vehicle. "<<endl;
    }
};
class Car : public Vehicle {
    public:
    void start(){
        cout<<"Car  is now started. "<<endl;
    }
};
class ElectricCar : public Car {
    public:
    void charge(){
        cout<<"Electric Car is charging.... "<<endl;
    }
};
int main(){
    ElectricCar car1;
    car1.charge();
    car1.start();
    car1.drive();
    return 0;
}
```

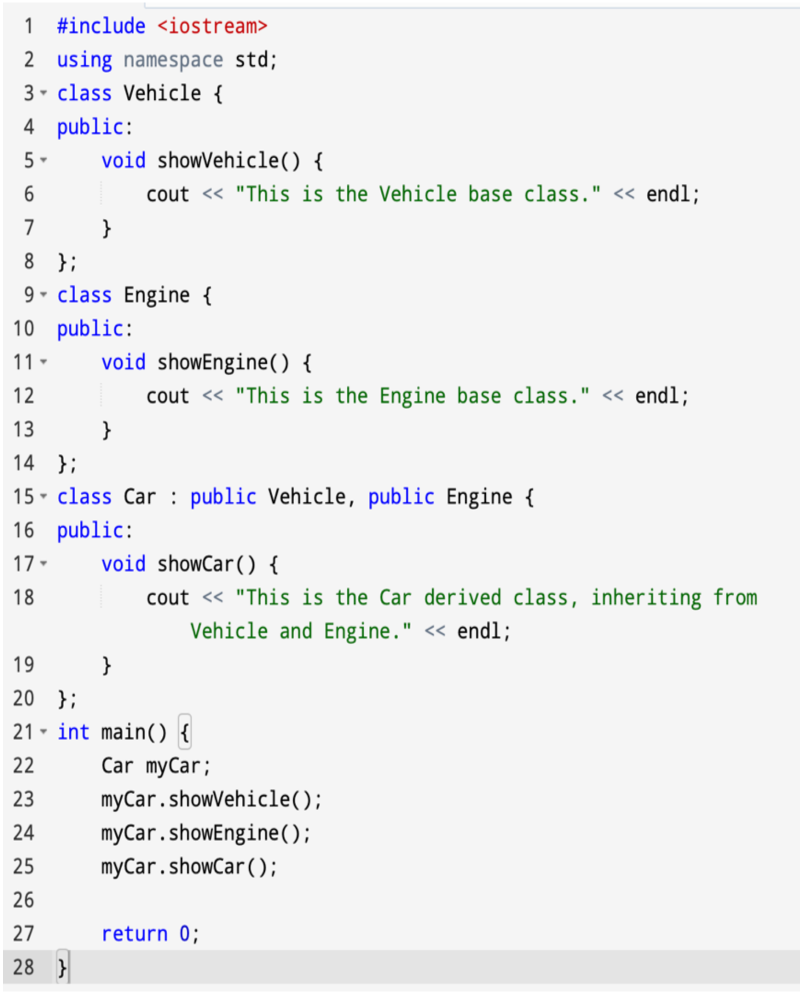


Output

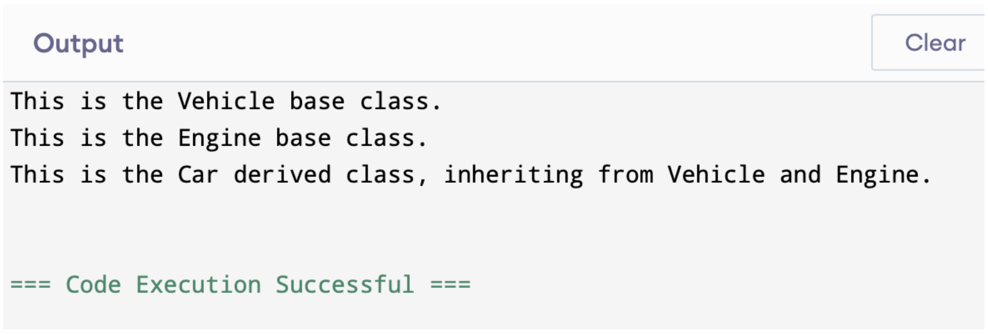
```
Electric Car is charging....
Car  is now started.
Driving a vehicle.


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

Exercise 5: **Hybrid Inheritance**

- Create a base class Vehicle and a base class Engine.
- Create a derived class Car that inherits from both Vehicle and Engine.
- Implement a main() function to demonstrate the usage of these classes



```cpp
#include <iostream>
using namespace std;
class Vehicle {
public:
    void showVehicle() {
        cout << "This is the Vehicle base class." << endl;
    }
};
class Engine {
public:
    void showEngine() {
        cout << "This is the Engine base class." << endl;
    }
};
class Car : public Vehicle, public Engine {
public:
    void showCar() {
        cout << "This is the Car derived class, inheriting from
            Vehicle and Engine." << endl;
    }
};
int main() {
    Car myCar;
    myCar.showVehicle();
    myCar.showEngine();
    myCar.showCar();

    return 0;
}
```



```
This is the Vehicle base class.
This is the Engine base class.
This is the Car derived class, inheriting from Vehicle and Engine.


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

DISCUSSION

In this lab, we explored the fundamental concept of inheritance in C++, which allows a class to acquire properties and behaviour's (data members and member functions) from another class. By implementing various types of inheritance—single, multiple, multilevel, hierarchical, and hybrid and we observed how inheritance facilitates code reusability and enhances the organization of classes in an object-oriented program.

CONCLUSION

In conclusion, the lab successfully demonstrated the concept and types of inheritance in C++. We learned how inheritance allows the creation of hierarchical class structures, leading to reusable, extensible, and well-organized code. By implementing different inheritance types, we understood the practical applications and potential challenges, such as ambiguity in multiple inheritance. This lab reinforced the significance of inheritance as a core pillar of object-oriented programming in C++. The knowledge gained will be instrumental in designing complex software systems that require efficient code reuse and logical class relationships.