



Himalaya College of Engineering

## **Advanced C++ Programming Lab Report**

Lab 4: TYPES OF INHERITANCE AND FUNCTION OVERRIDING

**Prepared By:** Ankit Belbase(HCE081BEI007)

**Subject:** Object-Oriented Programming (OOP) **Program:**

Bachelor of Electronics Engineering

**Institution:** Himalaya College of Engineering

**Date:** June 22,2025

## Objective:

1. To understand and implement various types of inheritance in C++ through practical examples.
2. To understand and implement virtual functions in C++ to achieve runtime polymorphism.

## Overview:

Inheritance is a fundamental concept in object-oriented programming that allows one class to inherit properties and methods from another class. In C++, inheritance is used to create a new class that is based on an existing class.

## Types of Inheritance:

1. **Single Inheritance:** A class (derived class) inherits from one base class.
2. **Multiple Inheritance:** A class inherits from more than one base class.
3. **Hierarchical Inheritance:** Multiple classes inherit from a single base class.
4. **Multilevel Inheritance:** A class inherits from another derived class.
5. **Hybrid Inheritance:** A combination of two or more types of inheritance.

## Virtual Functions:

Virtual functions in C++ allow derived classes to override methods defined in base classes. When a method is declared as `virtual`, C++ uses dynamic binding to determine which method to call at runtime based on the actual object type.

## Lab Questions for Inheritance:

### Ques1:

Create a base class `Shape` with a method `display` and create a derived class `Circle` that inherits from `Shape` and has an additional method `draw()` and implement in main function

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

class Shape {
public:
    void display() {
        cout << "This is a shape." << endl;
    }
};

class Circle : public Shape {
public:
    void draw() {
        cout << "Drawing a circle." << endl;
    }
}
```

```
};  
int main() {  
    Circle c;  
    c.display();  
    c.draw();  
    return 0;  
}
```

## OUTPUT:

```
This is a shape.  
Drawing a circle.
```

### Ques 2:

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

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

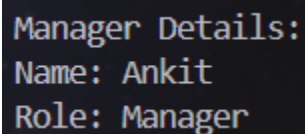
class Person {
public:
    void name() {
        cout << "Name: Ankit" << endl;
    }
};

class Employee {
public:
    void role() {
        cout << "Role: Manager" << endl;
    }
};

class Manager : public Person, public Employee {
public:
    void details() {
        cout << "Manager Details:" << endl;
        name();
        role();
    }
};

int main() {
    Manager m;
    m.details();
    return 0;
}
```

### OUTPUT:

A screenshot of a terminal window showing the output of the C++ program. The text is displayed in a light blue font on a dark background. It shows 'Manager Details:' followed by 'Name: Ankit' and 'Role: Manager' on separate lines.

```
Manager Details:
Name: Ankit
Role: Manager
```

### Ques 3:

1. Create a base class **Animal** with a method `speak()`.
2. Create two derived classes **Dog** and **Cat** that inherit from **Animal** and have their own `speak()` methods.
3. Implement a `main()` function to demonstrate the usage of these classes.

```

#include <iostream>
using namespace std;

class Animal {
public:
    void speak() {
        cout << "Animal speaks." << endl;
    }
};

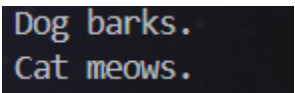
class Dog : public Animal {
public:
    void speak() {
        cout << "Dog barks." << endl;
    }
};

class Cat : public Animal {
public:
    void speak() {
        cout << "Cat meows." << endl;
    }
};

int main() {
    Dog d;
    Cat c;
    d.speak();
    c.speak();
    return 0;
}

```

## OUTPUT:



```

Dog barks.
Cat meows.

```

## Ques 4:

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

```

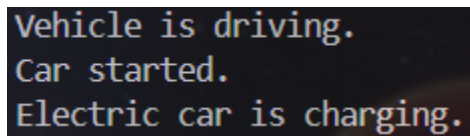
#include <iostream>
using namespace std;

class Vehicle {
public:
    void drive() {
        cout << "Vehicle is driving." << endl;
    }
};

```

```
class Car : public Vehicle { public: void start() {  
    cout << "Car started." << endl;  
}  
};  
  
class ElectricCar : public Car {  
public:  
    void charge() {  
        cout << "Electric car is charging." << endl;  
    }  
};  
  
int main() {  
    ElectricCar eCar;  
    eCar.drive();  
    eCar.start();  
    eCar.charge();  
    return 0;  
}
```

**OUTPUT:**

A screenshot of a terminal window showing the output of the program. The text is displayed on three separate lines: "Vehicle is driving.", "Car started.", and "Electric car is charging." The background of the terminal is dark, and the text is light-colored.

Vehicle is driving.  
Car started.  
Electric car is charging.

### Ques 5:

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

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

class Vehicle {
public:
    void drive() {
        cout << "Vehicle is driving." << endl;
    }
};

class Engine {
public:
    void start() {
        cout << "Engine started." << endl;
    }
};

class Car : public Vehicle, public Engine {
public:
    void run() {
        start();
        drive();
    }
};

int main() {
    Car c;
    c.run();
    return 0;
}
```

### OUTPUT:

```
Engine started.
Vehicle is driving.
```

## Lab Questions for virtual functions:

### Ques 1:

1. Create a base class **Base** with a virtual method `display()`.
2. Create a derived class **Derived** that overrides the `display()` method.
3. Implement a `main()` function where you create a **Base** pointer pointing to a **Derived** object and call the `display()` method.

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

class Base {
public:
    virtual void display() {
        cout << "Base class display method" << endl;
    }
};

class Derived : public Base {
public:
    void display() override {
        cout << "Derived class display method" << endl;
    }
};

int main() {
    Base* b;
    Derived d;
    b = &d;
    b->display();
    return 0;
}
```

### OUTPUT:

```
Derived class display method
```

### Ques 2:

1. Create a base class **Person** with a virtual method `show()`.
2. Create another base class **Employee** with a virtual method `show()`.
3. Create a derived class **Manager** that inherits from both **Person** and **Employee**, and overrides the `show()` method.
4. Implement a `main()` function to demonstrate the usage of these classes.

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

class Person {
public:
    virtual void show() {
        cout << "Person's show method" << endl;
    }
};
```

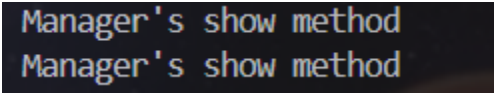


```
class Employee {
public:
    virtual void show() {
        cout << "Employee's show method" << endl;
    }
};

class Manager : public Person, public Employee {
public:
    void show() override {
        cout << "Manager's show method" << endl;
    }
};

int main() {
    Manager m;
    Person* p = &m;
    Employee* e = &m;
    p->show();
    e->show();
    return 0;
}
```

**OUTPUT:**



```
Manager's show method
Manager's show method
```

## **DISCUSSION:**

Inheritance in C++ allows one class to reuse the properties and methods of another, making code more organized and reusable. It supports hierarchical relationships between classes. Virtual functions enable runtime polymorphism, allowing the correct method to be called based on the actual object type, even when accessed through a base class pointer. Together, these concepts make C++ programs more flexible and maintainable.

## **CONCLUSION:**

The implementation of inheritance and virtual functions in C++ allows developers to build modular, maintainable, and reusable code. Inheritance simplifies code by promoting the reuse of existing functionality, while virtual functions enable polymorphism, ensuring that the appropriate behavior is executed for derived class objects. Through this lab, we gained practical experience in applying these concepts, reinforcing how they contribute to the power and flexibility of object-oriented programming in C++.