

Objectives:

- To understand and implement various types of inheritance in C++.
- To explore how function overriding and virtual functions enable polymorphism.
- To apply concepts of runtime polymorphism using base class pointers.

Tools and Libraries Used:

- Programming Language: C++
- IDE: Code::Blocks
- Libraries: `include <iostream>`, `include <string>`

Theory:

In C++, inheritance is a fundamental feature of object-oriented programming (OOP) that enables a new class, known as the derived class, to acquire the properties and behaviors (i.e., data members and member functions) of an existing class, known as the base class. This promotes code reusability, supports hierarchical classification, and facilitates a more natural representation of real-world relationships in code.

Inheritance allows a derived class not only to reuse the functionality of a base class but also to extend or customize it. The derived class inherits all accessible (typically public and protected) attributes and methods from the base class. Additionally, it can introduce new members of its own or override base class methods to implement specialized behavior.

Types of Inheritance in C++

1. Single Inheritance

A derived class inherits from a single base class.

Example:

```
class A {};
```

```
class B : public A {};
```

2. Multiple Inheritance

A derived class inherits from more than one base class.

Example:

```
class A {};  
class B {};  
class C : public A, public B {};
```

3. Hierarchical Inheritance

Multiple derived classes inherit from a common base class.

Example:

```
class A {};  
class B : public A {};  
class C : public A {};
```

4. Multilevel Inheritance

A derived class becomes a base class for another class.

Example:

```
class A {};  
class B : public A {};  
class C : public B {};
```

5. Hybrid Inheritance

A combination of two or more types of inheritance (e.g., multiple + hierarchical).

This can introduce ambiguity (such as the diamond problem), which is typically resolved using virtual inheritance.

Example:

```
class A {};  
class B : virtual public A {};  
class C : virtual public A {};  
class D : public B, public C {};
```

Function Overriding

Function overriding in C++ occurs when a derived class provides a new implementation for a base class method that has the same name, return type, and parameters. This allows the derived class to define behavior that is specific to its context, effectively replacing the base class version when invoked through a derived object.

To enable runtime polymorphism, the method in the base class should be declared with the virtual keyword. This ensures that the correct function is called based on the actual type of object, not the type of the pointer or reference to it. Additionally, the overridden function must be accessible, typically marked as public in the base class.

Virtual Functions

Virtual functions are essential for achieving dynamic dispatch (or runtime polymorphism) in C++. When a base class declares a member function as virtual, it instructs the compiler to determine the appropriate function to call at runtime, rather than at compile time. This decision is based on the actual type of the object, not the type of the pointer or reference through which it is accessed.

This behavior enables polymorphic behavior in inheritance hierarchies, allowing derived classes to override base class methods and have those overridden methods correctly invoked even when using base class pointers or references.

Characteristics:

- Declared using the virtual keyword in the base class.
- Enable calling derived class methods through base class pointers or references.
- Must be overridden in the derived class to exhibit polymorphic behavior.
- Typically used with base class pointers or references.

Syntax:

```
class Base {  
    public: virtual void show();  
};
```

/*exercise 1: Single Inheritance

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

```
#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;
}
```

/*Exercise 2: Multiple Inheritance

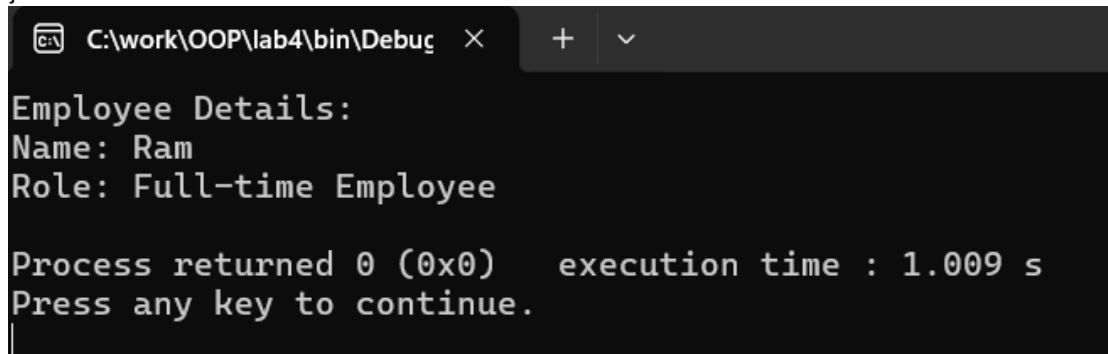
1. Create two base classes Person and Employee with appropriate methods.
2. Create a derived class Manager that inherits from both Person and Employee.
3. Implement a main() function to demonstrate the usage of these classes.*/*

```
#include <iostream>
using namespace std;
class Person {
public:
    void name() {
        cout << "Name: Ram" << endl;
    };
};
class Employee {
public:
    void role() {
        cout << "Role: Full-time Employee" << endl;
    };
};
class Manager: public Person, public Employee {
public:
    void details() {
```

```

cout << "Employee Details:" << endl;
name();
role();
});
int main() {
    Manager m;
    m.details();
}

```



The screenshot shows a debugger window titled "C:\work\OOP\lab4\bin\Debug" with a plus sign and a dropdown arrow. The output text is as follows:

```

Employee Details:
Name: Ram
Role: Full-time Employee

Process returned 0 (0x0)   execution time : 1.009 s
Press any key to continue.

```

/*Exercise 3: Hierarchical Inheritance

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;
}

```

```
C:\work\OOP\lab4\bin\Debug  X + v
Dog barks.
Cat meows.

Process returned 0 (0x0)   execution time : 1.018 s
Press any key to continue.
```

/*Exercise 4: Multilevel Inheritance

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();
}
```

```
C:\work\OOP\lab4\bin\Debug X + v
Vehicle is driving.
Car started.
Electric car is charging.

Process returned 0 (0x0)   execution time : 0.114 s
Press any key to continue.
```

/*Exercise 5: Hybrid Inheritance

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();
}
};
class ElectricCar: public Car{
public:
void electric(){
run();
}
};
int main() {
Car c;
ElectricCar ec;
cout<<"FOR NORMAL CAR"<<endl;
c.run();
cout<<"FOR ELECTRIC CAR"<<endl;
ec.electric();
return 0;}
```

```
C:\work\OOP\lab4\bin\Debug  X + v
FOR NORMAL CAR
Engine started.
Vehicle is driving.
FOR ELECTRIC CAR
Engine started.
Vehicle is driving.

Process returned 0 (0x0)    execution time : 1.017 s
Press any key to continue.
|
```