

OBJECTIVE

- To understand the concept of inheritance and its role in Object-Oriented Programming (OOP).
- To implement various types of inheritance such as single, multiple, and multilevel inheritance.

BACKGROUND THEORY

Inheritance is one of the fundamental concepts of Object-Oriented Programming (OOP). It allows a new class (called the **derived class**) to acquire the properties and behaviors (data members and member functions) of an existing class (called the **base class**). This promotes code reusability and logical hierarchy in programming.

Purpose and Benefits:

- **Code Reusability:** Once a class is written, it can be reused by other classes through inheritance, avoiding redundant code.
- **Extensibility:** Existing code can be extended with new features without modifying the original class.
- **Maintainability:** Organized and hierarchical structure makes code easier to maintain.
- **Polymorphism Support:** Inheritance works hand in hand with polymorphism for dynamic behavior at runtime.

Types of Inheritance in C++:

1. **Single Inheritance:** A derived class inherits from one base class.

```
class A {  
public:  
    void display() { cout << "Base class A"; }  
};  
class B : public A {  
};
```

2. **Multilevel Inheritance:** A class is derived from a derived class.

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

3. **Multiple Inheritance:** A class inherits from more than one base class.

```
class A { };
```

```
class B { };
```

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

4. **Hierarchical Inheritance:** Multiple derived classes inherit from a single base class.

5. **Hybrid Inheritance:** A combination of two or more types of inheritance.

Access Specifiers in Inheritance:

The type of inheritance (public, protected, or private) affects the accessibility of base class members in the derived class:

- **Public inheritance:** Public and protected members of the base class remain public and protected in the derived class.
- **Protected inheritance:** Public and protected members become protected in the derived class.
- **Private inheritance:** All inherited members become private in the derived class.

Example:

```
class Animal {
```

```
public:
```

```
    void eat() {
```

```
        cout << "Eating..." << endl;
```

```
    }
```

```
};
```

```
class Dog : public Animal {
```

```
public:
```

```
    void bark() {
```

```
        cout << "Barking..." << endl;
```

```
    }
```

```
};
```

Here, the class Dog inherits the eat() function from the base class Animal and also defines its own function bark().

Virtual Base Class (in Multiple Inheritance):

When multiple paths to a base class exist, ambiguity may arise. This is resolved using the virtual keyword.

cpp

CopyEdit

```
class A { };
```

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

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

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

Operator Overloading is one of the most important features of Object-Oriented Programming in C++. It allows us to redefine the way operators work for user-defined data types (like classes and structures). By overloading operators, we can perform operations such as addition, subtraction, comparison, etc., on objects as if they were built-in data types.

1. **Create a base class Shape** with a method display(), create a derived class Circle that inherits from Shape and has an additional method draw() and 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() {
```

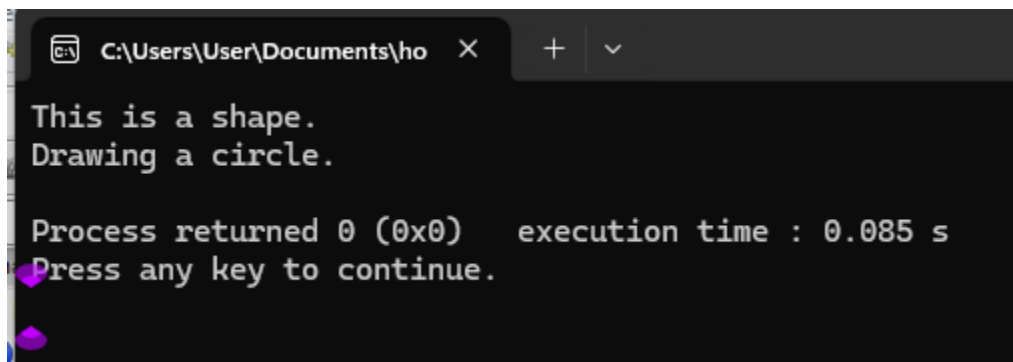
```

cout << "Drawing a circle." << endl;
}
};

int main() {
    Circle c;
    c.display(); // Inherited from Shape
    c.draw(); // Defined in Circle
    return 0;
}

```

Output:



```

C:\Users\User\Documents\ho
This is a shape.
Drawing a circle.

Process returned 0 (0x0)   execution time : 0.085 s
Press any key to continue.

```

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

```

#include <iostream>

using namespace std;

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

class Employee {

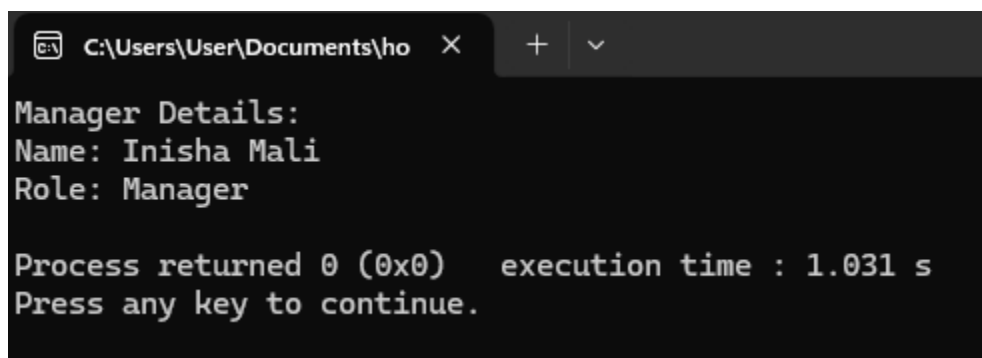
```

```
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



```
C:\Users\User\Documents\ho  X  +  v
Manager Details:
Name: Inisha Mali
Role: Manager

Process returned 0 (0x0)   execution time : 1.031 s
Press any key to continue.
```

Discussion

In this lab, we explored the concept of inheritance in C++, a fundamental principle of object-oriented programming that promotes code reusability and hierarchical relationships among classes. By creating base and derived classes, we understood how members and methods can be shared and extended.

Conclusion

The lab on inheritance helped us understand how to build structured and scalable programs using class hierarchies. Inheritance not only reduces code redundancy but also allows for flexible and maintainable designs. By practicing inheritance through hands-on coding, we gained practical insight into how real-world relationships can be modeled in object-oriented programming.