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: G++
- Libraries: include <iostream>, include <string>

Theory:

In C++, inheritance is a powerful feature of object-oriented programming (OOP) that allows a new class (called the derived class) to acquire the properties and behaviors of an existing class (called the base class). This promotes code reusability, hierarchical classification, and a more natural mapping of real-world entities into code.

Inheritance not only enables one class to reuse the functionality of another but also allows programmers to extend and customize the existing behaviors. The derived class inherits accessible attributes and methods of the base class, and it can also define its own members or override existing ones.

Types of Inheritance in C++

1. Single Inheritance:

   A derived class inherits from a single base class.

   Example:
   ```
   class Circle : public Shape
   ```
2. Multiple Inheritance:

   A derived class inherits from more than one base class.

   Example:
   ```
   class Manager : public Person, public Employee
   ```

3. Hierarchical Inheritance:

   Multiple derived classes inherit from a single base class.

   Example: `class Dog : public Animal, class Cat : public Animal`

4. Multilevel Inheritance:

   A derived class is itself used as a base class for another class.

   Example: `class ElectricCar : public Car : public Vehicle`

5. Hybrid Inheritance:

   A combination of two or more types of inheritance. This often leads to ambiguity and requires careful handling using virtual inheritance.

Function Overriding

Function overriding occurs when a derived class redefines a base class method with the same name, return type, and parameters. This allows the derived class to provide its own specific implementation of the method.

- The overridden function in the base class must be accessible (usually public) and is commonly declared as virtual to enable runtime polymorphism.

Virtual Functions

Virtual functions are key to achieving dynamic dispatch in C++. When a base class declares a method as virtual, it tells the compiler to wait until runtime to resolve which class's version of the method should be called—based on the type of object, not the pointer/reference.

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(); // virtual function
};
```

Lab Questions:

Q no 1:

Create a base class Shape with a method display().

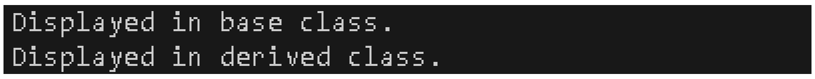
Create a derived class Circle that inherits from Shape and has an additional method draw().

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

Code:

```
 #include<iostream>
using namespace std;
class shape {
    public:
    void display()
    {
        cout<<"Displayed in base class."<<endl;
    }
};
class circle:public shape {
    public:
    void draw()
    {
        cout<<"Displayed in derived class."<<endl;
    }
};
int main() {
    circle c;
    c.display();
    c.draw();
    return 0;
}
```

Output:



```
Displayed in base class.
Displayed in derived class.
```

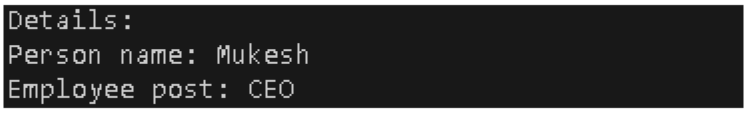
Q no 2:

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.

Code:

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

Output:



```
Details:
Person name: Mukesh
Employee post: CEO
```

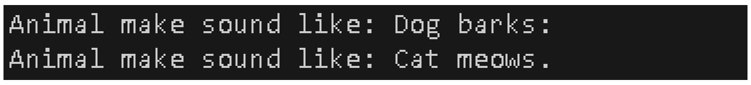
Q no 3:

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.

Code:

```
 #include<iostream>
using namespace std;
class Animal {
    public:
    void speak1() {
        cout<<"Animal make sound like: ";
    }
};
class Dog : public Animal {
       public:
       void speak(){
           speak1();
           cout<<"Dog barks: "<<endl;
       }
   };
   class Cat : public Animal {
       public:
       void speak(){
           speak1();
           cout<<"Cat meows. "<<endl;
       }
   };
   int main()
   {
       Dog D;
       Cat C;
       D.speak();
       C.speak();
       return 0;
   }
```

Output:



```
Animal make sound like: Dog barks:
Animal make sound like: Cat meows.
```

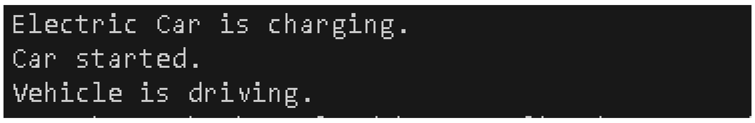
Q no 4:

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.

Code:

```
 #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 car1;
       car1.charge();
       car1.start();
       car1.drive();
       return 0;
   }
```

Output:



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

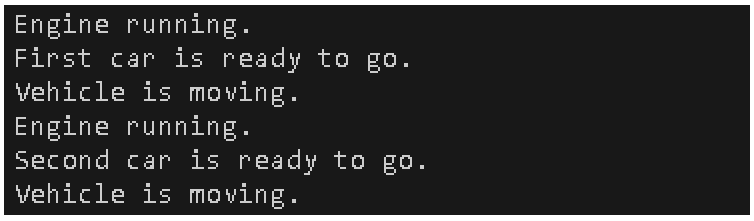
Q no 5:

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.

Code:

```
 #include<iostream>
using namespace std;
class Vehicle {
    public:
    void move (){
        cout<<"Vehicle is moving. "<<endl;
    }
};
class Engine {
       public:
       void run (){
           cout<<"Engine running. "<<endl;
       }
   };
   class Car1 : public Vehicle, public Engine {
       public:
       void ready(){
           run();
           cout<<"First car is ready to go. "<<endl;
           move();
       }
   };
   class Car2 : public Vehicle, public Engine {
       public:
       void ready(){
           run();
           cout<<"Second car is ready to go. "<<endl;
           move();
       }
   };
   int main(){
       Car1 c1;
       Car2 c2;
       c1.ready();
       c2.ready();
       return 0;
   }
```

Output:



```
Engine running.
First car is ready to go.
Vehicle is moving.
Engine running.
Second car is ready to go.
Vehicle is moving.
```

Conclusion:

This lab demonstrated the concept of inheritance and polymorphism in C++. Various types of inheritance—single, multiple, hierarchical, multilevel, and hybrid—were explored through practical examples. The implementation of function overriding and virtual functions illustrated how polymorphism enables flexible and reusable code. Overall, the exercises reinforced the principles of object-oriented programming by showing how derived classes can extend and customize the behavior of base classes effectively.