Types of inheritance

Multiple Inheritance in C++

Multiple inheritance means a class can inherit from **more than one base class**. It allows you to combine functionalities from multiple sources.

```
class Derived : public Base1, public Base2 {
   // Derived class code
};
```

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

class Person {
public:
   void displayPerson() {
      cout << "I am a person." << endl;
   }
};

class Employee {
public:
   void displayEmployee() {
      cout << "I am an employee." << endl;
   }
};

class Manager : public Person, public Employee {
public:
   void displayManager() {
      cout << "I am a manager." << endl;
   }
};

int main() {
   Manager m;
   m.displayPerson();   // From Person
   m.displayEmployee(); // From Employee
   m.displayManager();  // Own function
   return 0;
}
```

```
I am a person.
I am an employee.
I am a manager.
```

Ambiguity Problem in Multiple Inheritance

If both base classes have a function with the **same name**, it causes ambiguity in the derived class.

```
class A {
public:
   void greet() { cout << "Hello from A" << endl; }
};

class B {
public:
   void greet() { cout << "Hello from B" << endl; }
};

class C : public A, public B {
public:
   void sayHello() {
      // greet(); // Error: Ambiguous
      A::greet(); // Resolves ambiguity
      B::greet(); //
   }
};
```

What is Multilevel Inheritance?

Multilevel Inheritance means a class is **derived from a class**, which is **already derived from another class**.

- Think of it like a family tree:
- Grandparent → Parent → Child

```
// Base class
class Animal {
public:
   void eat() {
      cout << "Animal eats food." << endl;
   }
};

// Derived from Animal
class Mammal : public Animal {
public:
   void walk() {
      cout << "Mammal walks." << endl;
   }
};

// Derived from Mammal
class Dog : public Mammal {
public:
   void bark() {
      cout << "Dog barks." << endl;
   }
};

int main() {
   Dog d;
   d.eat();   // From Animal
   d.walk();  // From Mammal
   d.bark();  // From Dog
   return 0;
}
```

```
Animal eats food.
Mammal walks.
Dog barks.
```

What is Hierarchical Inheritance?

Hierarchical inheritance means **multiple derived classes inherit from a single base class**.

- Think: **One parent, many children.**
- It's like a root class that shares common traits with several branches.

```
// Base class
class Animal {
public:
   void eat() {
      cout << "Animal eats food." << endl;
   }
};

// Derived class 1
class Dog : public Animal {
public:
   void bark() {
      cout << "Dog barks." << endl;
   }
};

// Derived class 2
class Cat : public Animal {
public:
   void meow() {
      cout << "Cat meows." << endl;
   }
};
```

```
int main() {
   Dog d;
   Cat c;

   d.eat();   // Inherited from Animal
   d.bark();  // Own method

   c.eat();   // Inherited from Animal
   c.meow();  // Own method

   return 0;
}
```

Why use Hierarchical Inheritance?

- **Reuse** common functionality (like eat()) across multiple derived classes.
- Helps organize code into clean, logical categories.
- A great fit when multiple classes share a base behavior, but also need **unique features.**

Hybrid Inheritance in C++?

Hybrid inheritance is a combination of **two or more types of inheritance** (like multiple + multilevel, or hierarchical + multiple).
It often leads to the **diamond problem**

- Person is a base class.
- Employee and Student both inherit from Person.
- WorkingStudent inherits from both Employee and Student.
- So it's a mix of **hierarchical + multiple** inheritance.

```
class Person {
public:
   void info() {
      cout << "I am a person." << endl;
   }
};

class Employee : public Person { };
class Student : public Person { };

// Hybrid inheritance
class WorkingStudent : public Employee, public Student {
   // Uh oh! Two copies of Person in here.
};
```

```
   WorkingStudent ws;
   ws.info(); // ERROR: which info()? From Employee or Student?
```

Solution: Virtual Inheritance

- To fix the diamond problem, use virtual inheritance:

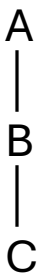
```
class Person {
public:
   void info() {
      cout << "I am a person." << endl;
   }
};

class Employee : virtual public Person { };
class Student : virtual public Person { };

class WorkingStudent : public Employee, public Student {
   // Now only ONE shared Person base
};


int main() {
   WorkingStudent ws;
   ws.info();  // No ambiguity
   return 0;
}
```

I am a person.

Inheritance Type	Use case
Single	One-to-one relationship
Multilevel	Inherit in a chain
Hierarchical	One base, many derived
Multiple	One derived, many bases
Hybrid	Real-world complex structures

1. **Single Inheritance**

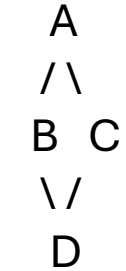
2. **Multilevel Inheritance**



3. **Hierarchical Inheritance**

4. **Multiple Inheritance**

5. **Hybrid Inheritance (Diamond Problem)**



- Person → base class (name, age)
- Student and Professor → both inherit from Person
- TA (Teaching Assistant) → inherits from both Student and Professor
  (*Hybrid Inheritance!*)

```
// Base class
class Person {
public:
    string name;
    int age;
    void setInfo(string n, int a) {
        name = n;
        age = a;
    }
    void showInfo() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};


        int main() {
            TA ta;
            ta.setInfo("Alex", 24);
            ta.showInfo();
            ta.study();
            ta.teach();
            ta.work();
            return 0;
        }
```

```
// Student and Professor use virtual inheritance
class Student : virtual public Person {
public:
    void study() {
        cout << name << " is studying." << endl;
    }
};

class Professor : virtual public Person {
public:
    void teach() {
        cout << name << " is teaching." << endl;
    }
};

// Hybrid Inheritance
class TA : public Student, public Professor {
public:
    void work() {
        cout << name << " is working as a TA." << endl;
    }
};
```