

## is-a relationship

Inheritance is an **is-a relationship**. We use inheritance only if an **is-a relationship** is present between the two classes.

Here are some examples:

* A car is a vehicle.
* Orange is a fruit.
* A surgeon is a doctor.
* A dog is an animal.

## **Purpose of Inheritance in C++**

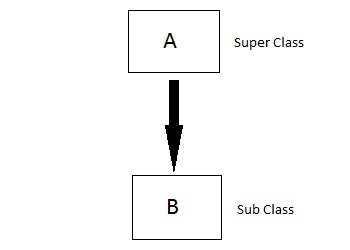
1. Code Reusability
2. Method Overriding (Hence, Runtime Polymorphism.)
3. Use of Virtual Keyword

Types of Inheritance in C++

1. Single Inheritance
2. Multiple Inheritance
3. Hierarchical Inheritance
4. Multilevel Inheritance
5. Hybrid Inheritance (also known as Virtual Inheritance)

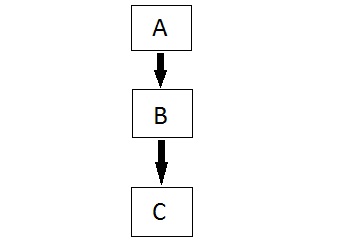
## **Single Inheritance in C++**

In this type of inheritance one derived class inherits from only one base class. It is the most simplest form of Inheritance.



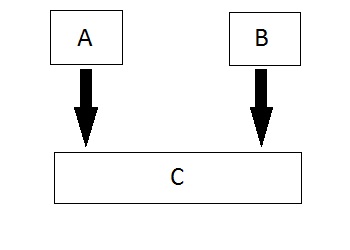
## **Multilevel Inheritance in C++**

In this type of inheritance the derived class inherits from a class, which in turn inherits from some other class. The Super class for one, is sub class for the other.



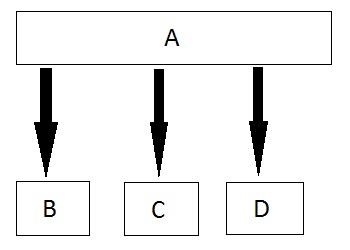
## **Multiple Inheritance in C++**

In this type of inheritance a single derived class may inherit from two or more than two base classes.



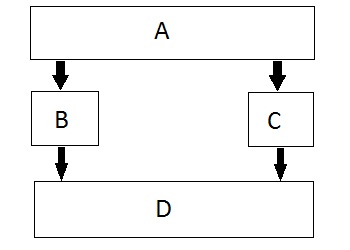
## **Hierarchical Inheritance in C++**

In this type of inheritance, multiple derived classes inherits from a single base class.



## **Hybrid (Virtual) Inheritance in C++**

Hybrid Inheritance is combination of Hierarchical and Mutilevel Inheritance.



// Base class  
class Vehicle {  
  public:  
    string brand = "Ford";  
    void honk() {  
      cout << "Tuut, tuut! \n" ;  
    }  
};  
  
// Derived class  
**class Car: public Vehicle** {  
  public:  
    string model = "Mustang";  
};  
  
int main() {  
  Car myCar;  
  myCar.honk();  
  cout << myCar.brand + " " + myCar.model;  
  return 0;  
}

OUTPUT:  
Tuut, tuut!  
Ford Mustang

## **Multilevel Inheritance**

A class can also be derived from one class, which is already derived from another class.

In the following example, MyGrandChild is derived from class MyChild (which is derived from MyClass).

### **Example**

// Base class (parent)  
class MyClass {  
  public:  
    void myFunction() {  
      cout << "Some content in parent class." ;  
    }  
};  
  
// Derived class (child)  
class MyChild: public MyClass {  
};  
  
// Derived class (grandchild)  
class MyGrandChild: public MyChild {  
};  
  
int main() {  
  MyGrandChild myObj;  
  myObj.myFunction();  
  return 0;  
}

## **Multiple Inheritance**

A class can also be derived from more than one base class, using a **comma-separated list:**

### **Example**

// Base class  
class MyClass {  
  public:  
    void myFunction() {  
      cout << "Some content in parent class." ;  
    }  
};  
  
// Another base class  
class MyOtherClass {  
  public:  
    void myOtherFunction() {  
      cout << "Some content in another class." ;  
    }  
};  
  
// Derived class  
**class MyChildClass: public MyClass, public MyOtherClass** {  
};  
  
int main() {  
  MyChildClass myObj;  
  myObj.myFunction();  
  myObj.myOtherFunction();  
  return 0;  
}

## **Access Specifiers**

You learned from the [Access Specifiers](https://www.w3schools.com/cpp/cpp_access_specifiers.asp) chapter that there are three specifiers available in C++. Until now, we have only used public (members of a class are accessible from outside the class) and private (members can only be accessed within the class). The third specifier, protected, is similar to private, but it can also be accessed in the **inherited** class:

### **Example**

// Base class  
class Employee {  
  **protected: // Protected access specifier**  
    int salary;  
};  
  
// Derived class  
class Programmer: public Employee {  
  public:  
    int bonus;  
    void setSalary(int s) {  
      salary = s;  
    }  
    int getSalary() {  
      return salary;  
    }  
};  
  
int main() {  
  Programmer myObj;  
  myObj.setSalary(50000);  
  myObj.bonus = 15000;  
  cout << "Salary: " << myObj.getSalary() << "\n";  
  cout << "Bonus: " << myObj.bonus << "\n";  
  return 0;  
}