# C++ Inheritance

In C++, inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically. In such way, you can reuse, extend or modify the attributes and behaviors which are defined in other class.

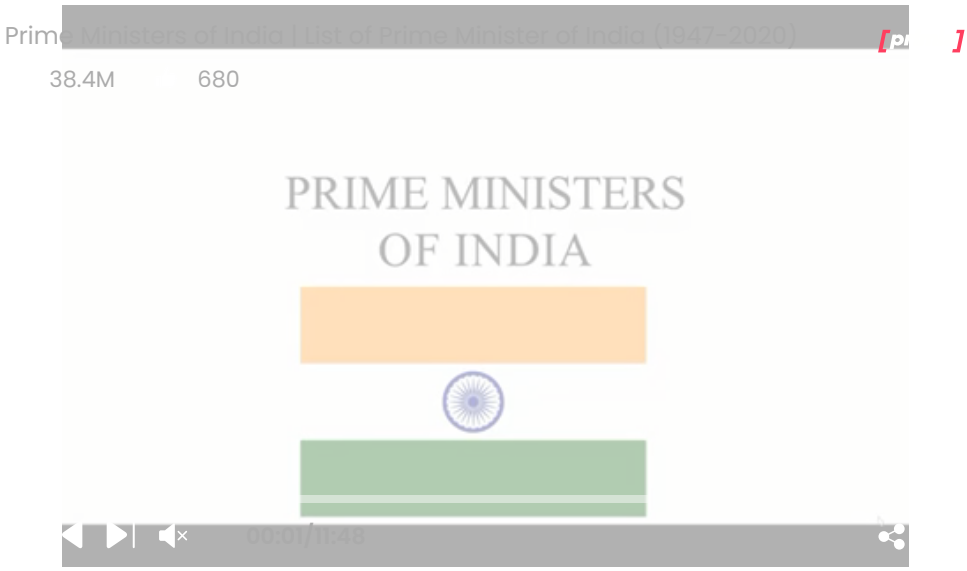
In C++, the class which inherits the members of another class is called derived class and the class whose members are inherited is called base class. The derived class is the specialized class for the base class.

## Advantage of C++ Inheritance

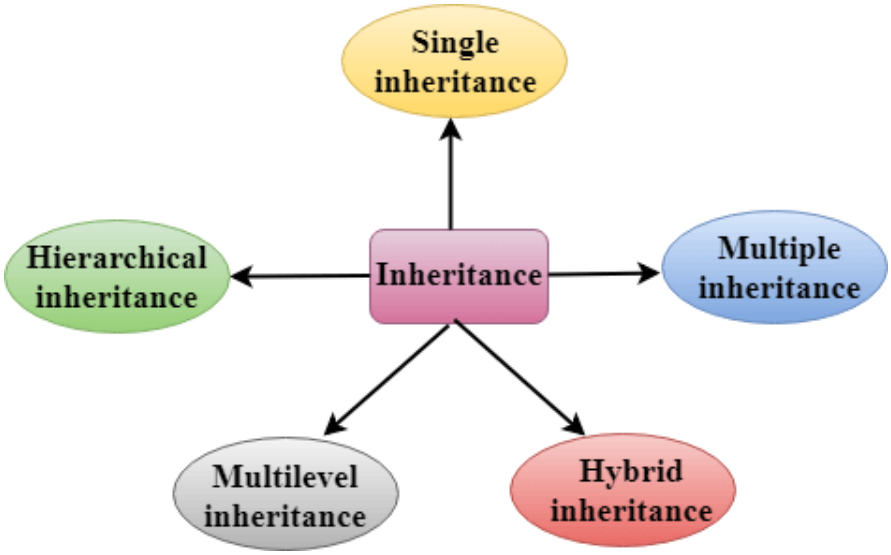
**Code reusability:** Now you can reuse the members of your parent class. So, there is no need to define the member again. So less code is required in the class.

## Types Of Inheritance

**C++ supports five types of inheritance:**




- Single inheritance
- Multiple inheritance
- Hierarchical inheritance
- Multilevel inheritance
- Hybrid inheritance




## Derived Classes

A Derived class is defined as the class derived from the base class.

The Syntax of Derived class:

```
class derived_class_name :: visibility-mode base_class_name
{
   // body of the derived class.
}
```

**Where,**

**derived_class_name:** It is the name of the derived class.

**visibility mode:** The visibility mode specifies whether the features of the base class are publicly inherited or privately inherited. It can be public or private.

**base_class_name:** It is the name of the base class.

- When the base class is privately inherited by the derived class, public members of the base class becomes the private members of the derived class. Therefore, the public members of the base class are not accessible by the objects of the derived class only by the member functions of the derived class.
- When the base class is publicly inherited by the derived class, public members of the base class also become the public members of the derived class. Therefore, the public members of the base class are accessible by the objects of the derived class as well as by the member functions of the base class.

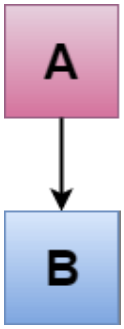
### Note:

- In C++, the default mode of visibility is private.

- The private members of the base class are never inherited.

## C++ Single Inheritance

**Single inheritance** is defined as the inheritance in which a derived class is inherited from the only one base class.




Where 'A' is the base class, and 'B' is the derived class.

## C++ Single Level Inheritance Example: Inheriting Fields

When one class inherits another class, it is known as single level inheritance. Let's see the example of single level inheritance which inherits the fields only.

```
#include <iostream>
using namespace std;
 class Account {
   public:
   float salary = 60000;
 };
   class Programmer: public Account {
   public:
   float bonus = 5000;
   };
```

```
int main(void) {
    Programmer p1;
    cout<<"Salary: "<<p1.salary<<endl;
    cout<<"Bonus: "<<p1.bonus<<endl;
   return 0;
}
```

Output:

```
Salary: 60000
Bonus: 5000
```

In the above example, Employee is the **base** class and Programmer is the **derived** class.

## C++ Single Level Inheritance Example: Inheriting Methods

Let's see another example of inheritance in C++ which inherits methods only.

```
#include <iostream>
using namespace std;
 class Animal {
  public:
 void eat() {
   cout<<"Eating..."<<endl;
 }
  };
  class Dog: public Animal
  {
     public:
    void bark(){
   cout<<"Barking...";
    }
  };
int main(void) {
   Dog d1;
   d1.eat();
   d1.bark();
   return 0;
}
```

Output:

```
Eating...
Barking...
```

Let's see a simple example.

```
#include <iostream>
using namespace std;
class A
{
   int a = 4;
   int b = 5;
   public:
   int mul()
   {
      int c = a*b;
      return c;
   }
};

class B : private A
{
   public:
   void display()
   {
      int result = mul();
      std::cout <<"Multiplication of a and b is : "<<result<< std::endl;
```

```
    }
};
int main()
{
   B b;
   b.display();

    return 0;
}
```

Output:

```
Multiplication of a and b is : 20
```

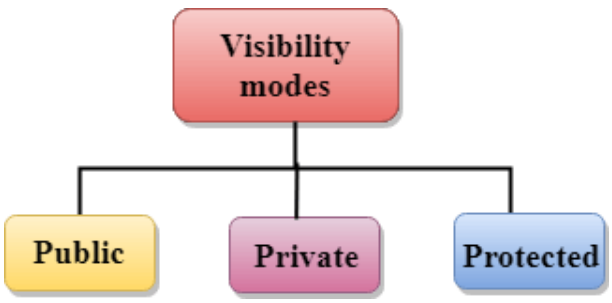
In the above example, class A is privately inherited. Therefore, the mul() function of class 'A' cannot be accessed by the object of class B. It can only be accessed by the member function of class B.

## How to make a Private Member Inheritable

The private member is not inheritable. If we modify the visibility mode by making it public, but this takes away the advantage of data hiding.

C++ introduces a third visibility modifier, i.e., **protected**. The member which is declared as protected will be accessible to all the member functions within the class as well as the class immediately derived from it.

**Visibility modes can be classified into three categories:**




- **Public**: When the member is declared as public, it is accessible to all the functions of the program.
- **Private**: When the member is declared as private, it is accessible within the class only.

- **Protected**: When the member is declared as protected, it is accessible within its own class as well as the class immediately derived from it.

## Visibility of Inherited Members

| Base class visibility | Derived class visibility | | |
|---|---|---|---|
| | Public | Private | Protected |
| Private | Not Inherited | Not Inherited | Not Inherited |
| Protected | Protected | Private | Protected |
| Public | Public | Private | Protected |

## C++ Multilevel Inheritance

**Multilevel inheritance** is a process of deriving a class from another derived class.


## C++ Multi Level Inheritance Example

When one class inherits another class which is further inherited by another class, it is known as multi level inheritance in C++. Inheritance is transitive so the last derived class acquires all the members of all its base classes.

Let's see the example of multi level inheritance in C++.

```
#include <iostream>
using namespace std;
 class Animal {
```

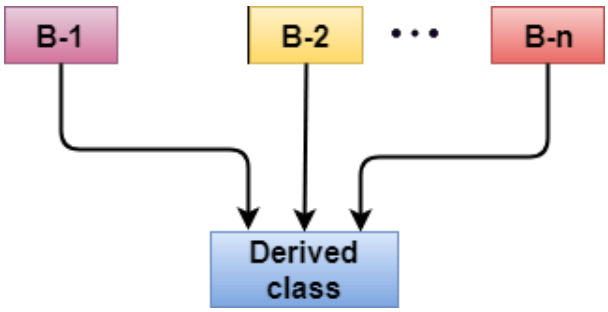
```
  public:
 void eat() {
   cout<<"Eating..."<<endl;
 }
  };
  class Dog: public Animal
  {
     public:
    void bark(){
   cout<<"Barking..."<<endl;
    }
  };
  class BabyDog: public Dog
  {
     public:
    void weep() {
   cout<<"Weeping...";
    }
  };
int main(void) {
   BabyDog d1;
   d1.eat();
   d1.bark();
    d1.weep();
    return 0;
}
```

Output:

```
Eating...
Barking...
Weeping...
```

## C++ Multiple Inheritance

**Multiple inheritance** is the process of deriving a new class that inherits the attributes from two or more classes.




**Syntax of the Derived class:**

```
class D : visibility B-1, visibility B-2, ?
{
   // Body of the class;
}
```

Let's see a simple example of multiple inheritance.

```
#include <iostream>
using namespace std;
class A
{
   protected:
    int a;
   public:
   void get_a(int n)
   {
       a = n;
   }
};
```

```
class B
{
   protected:
   int b;
   public:
   void get_b(int n)
   {
      b = n;
   }
};
class C : public A,public B
{
  public:
   void display()
   {
      std::cout << "The value of a is : " <<a<< std::endl;
      std::cout << "The value of b is : " <<b<< std::endl;
      cout<<"Addition of a and b is : "<<a+b;
   }
};
int main()
{
  C c;
  c.get_a(10);
  c.get_b(20);
  c.display();

   return 0;
}
```

Output:

```
The value of a is : 10
The value of b is : 20
Addition of a and b is : 30
```

In the above example, class 'C' inherits two base classes 'A' and 'B' in a public mode.

## Ambiquity Resolution in Inheritance

Ambiguity can be occurred in using the multiple inheritance when a function with the same name occurs in more than one base class.

Let's understand this through an example:

```
#include <iostream>
using namespace std;
class A
{
   public:
   void display()
   {
      std::cout << "Class A" << std::endl;
   }
};
class B
{
   public:
   void display()
   {
      std::cout << "Class B" << std::endl;
   }
};
class C : public A, public B
{
   void view()
   {
      display();
   }
};
int main()
{
   C c;
   c.display();
   return 0;
}
```

Output:

```
error: reference to 'display' is ambiguous
        display();
```

- The above issue can be resolved by using the class resolution operator with the function. In the above example, the derived class code can be rewritten as:

```
class C : public A, public B
{
   void view()
   {
      A :: display();       // Calling the display() function of class A.
      B :: display();       // Calling the display() function of class B.

   }
};
```

An ambiguity can also occur in single inheritance.

Consider the following situation:

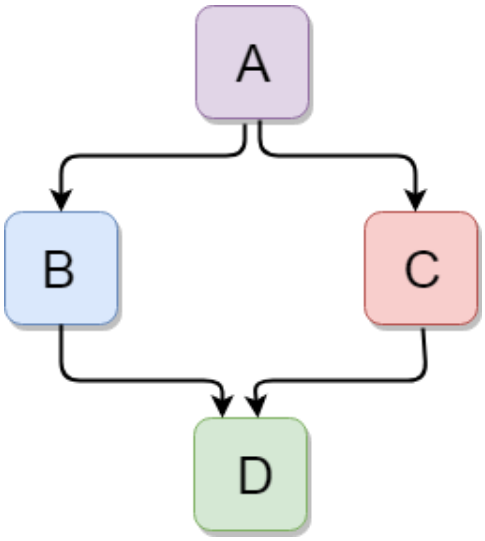
```
class A
{
  public:
void display()
{
  cout<<?Class A?;
}
};
class B
{
 public:
 void display()
{
 cout<<?Class B?;
}
};
```

In the above case, the function of the derived class overrides the method of the base class. Therefore, call to the display() function will simply call the function defined in the derived class. If we want to invoke the base class function, we can use the class resolution operator.

```
int main()
{
    B b;
   b.display();              // Calling the display() function of B class.
   b.B :: display();       // Calling the display() function defined in B class.
}
```

## C++ Hybrid Inheritance

Hybrid inheritance is a combination of more than one type of inheritance.




Let's see a simple example:

```
#include <iostream>
using namespace std;
class A
{
   protected:
   int a;
   public:
   void get_a()
   {
      std::cout << "Enter the value of 'a' : " << std::endl;
      cin>>a;
```

```
    }
};

class B : public A
{
    protected:
    int b;
    public:
    void get_b()
    {
        std::cout << "Enter the value of 'b' : " << std::endl;
       cin>>b;
    }
};
class C
{
    protected:
    int c;
    public:
    void get_c()
    {
        std::cout << "Enter the value of c is : " << std::endl;
        cin>>c;
    }
};

class D : public B, public C
{
    protected:
    int d;
    public:
    void mul()
    {
         get_a();
         get_b();
         get_c();
         std::cout << "Multiplication of a,b,c is : " <<a*b*c<< std::endl;
    }
```

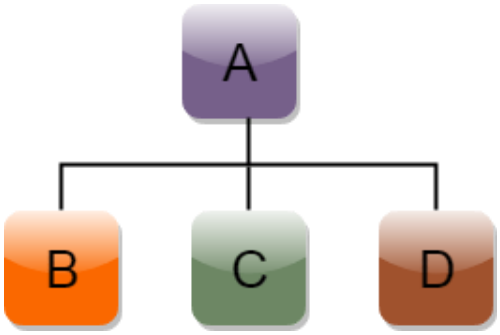
```
};
int main()
{
    D d;
    d.mul();
    return 0;
}
```

Output:

```
Enter the value of 'a' :
10
Enter the value of 'b' :
20
Enter the value of c is :
30
Multiplication of a,b,c is : 6000
```

## C++ Hierarchical Inheritance

Hierarchical inheritance is defined as the process of deriving more than one class from a base class.




**Syntax of Hierarchical inheritance:**

```
class A
{
   // body of the class A.
}
class B : public A
{
   // body of class B.
}
```

```
class C : public A
{
   // body of class C.
}
class D : public A
{
   // body of class D.
}
```

Let's see a simple example:

```
#include <iostream>
using namespace std;
class Shape              // Declaration of base class.
{
   public:
   int a;
   int b;
   void get_data(int n,int m)
   {
      a= n;
      b = m;
   }
};
class Rectangle : public Shape  // inheriting Shape class
{
   public:
   int rect_area()
   {
      int result = a*b;
      return result;
   }
};
class Triangle : public Shape    // inheriting Shape class
{
   public:
   int triangle_area()
   {
      float result = 0.5*a*b;
```

```
        return result;
    }
};
int main()
{
    Rectangle r;
    Triangle t;
    int length,breadth,base,height;
    std::cout << "Enter the length and breadth of a rectangle: " << std::endl;
    cin>>length>>breadth;
    r.get_data(length,breadth);
    int m = r.rect_area();
    std::cout << "Area of the rectangle is : " <<m<< std::endl;
    std::cout << "Enter the base and height of the triangle: " << std::endl;
    cin>>base>>height;
    t.get_data(base,height);
    float n = t.triangle_area();
    std::cout <<"Area of the triangle is : "  << n<<std::endl;
    return 0;
}
```

Output:

```
Enter the length and breadth of a rectangle:
23
20
Area of the rectangle is : 460
Enter the base and height of the triangle:
2
5
Area of the triangle is : 5
```

← Prev Next →