

Unit-2

Date: 1/4/2021

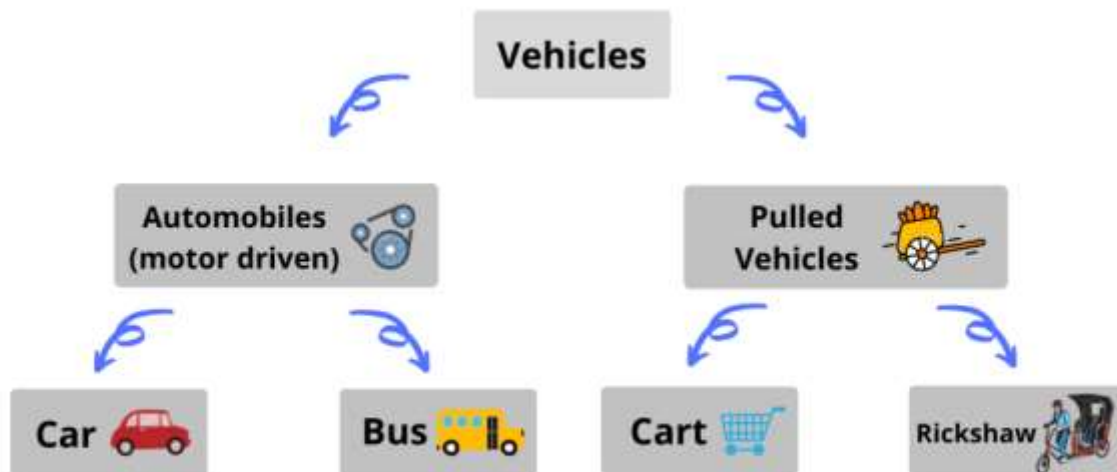
Chapter-1

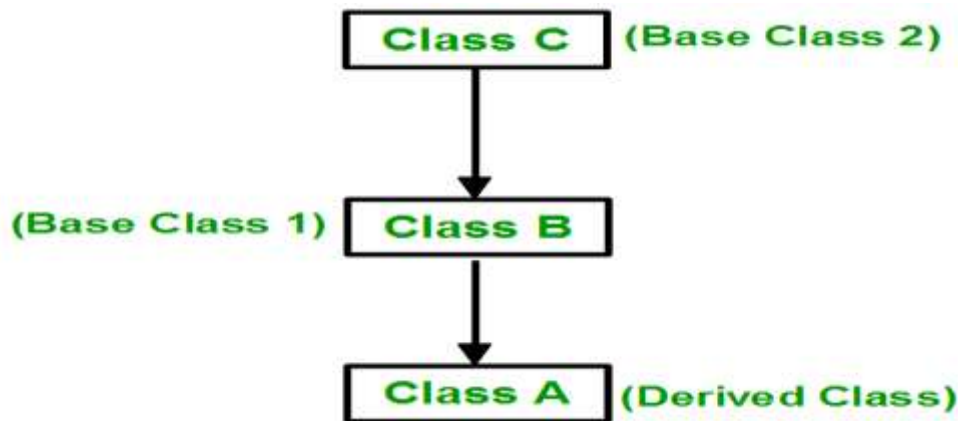
Reetu Bhardwaj

What is Inheritance?

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.

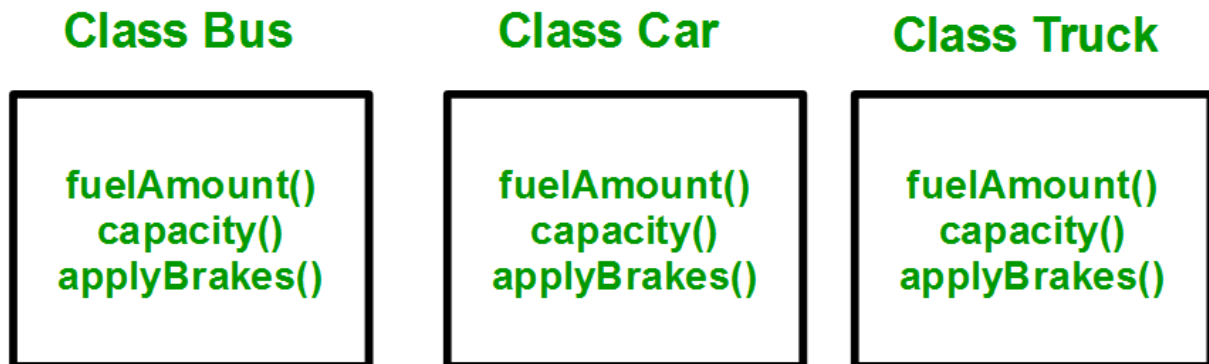
In C++, the class which inherits the members of another class is called derived class and the class whose members are inherited is called base class. The derived class is the specialized class for the base class.





Why and when to use inheritance?

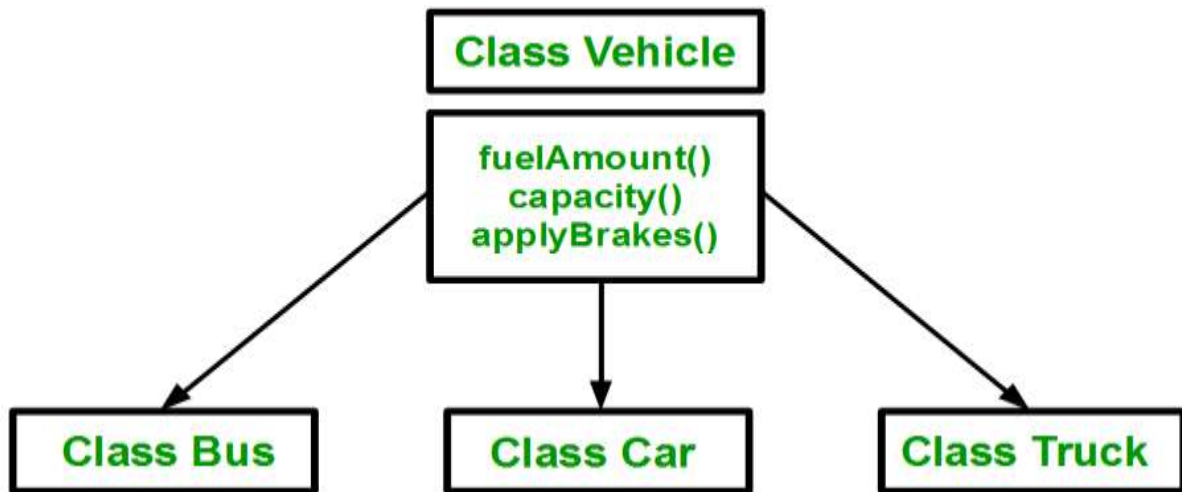
See the below diagram...



For classes Bus, Car and Truck, the methods `fuelAmount()`, `capacity()`, `applyBrakes()` will be same. If we create these classes avoiding inheritance then we have to write all of these functions in each of the three.

Above process results in duplication of same code 3 times. This increases the chances of error and data redundancy. To avoid this type of situation, inheritance is used.

Representation of same above class by using inheritance....



Advantages of Inheritance

- ❖ It allows the code to be reused as many times as needed.
- ❖ The base class once defined and once it is compiled, it need not be reworked.
- ❖ Saves time and effort as the main code need not be written again.

Syntax to implement Inheritance:

```
class subclass_name : access_mode base_class_name
{
    //body of subclass
};
```

Access Mode (Visibility of Inheritance)

There are three access modes in inheritance

❖ **Private**

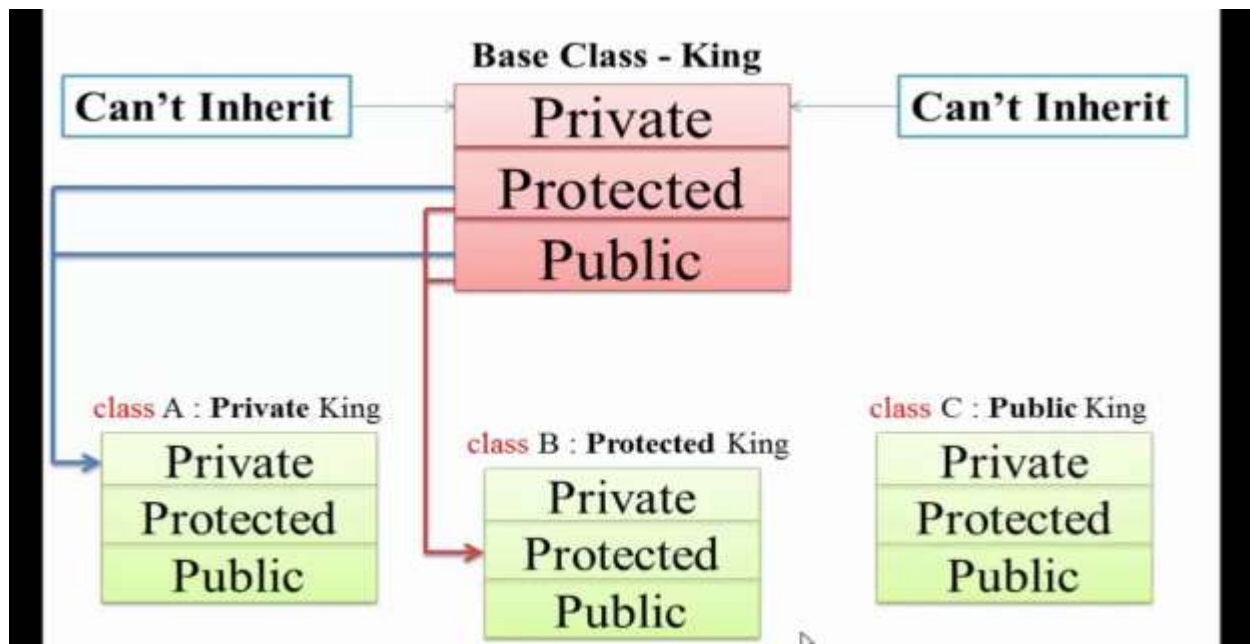
❖ **Public**

❖ **Protected**

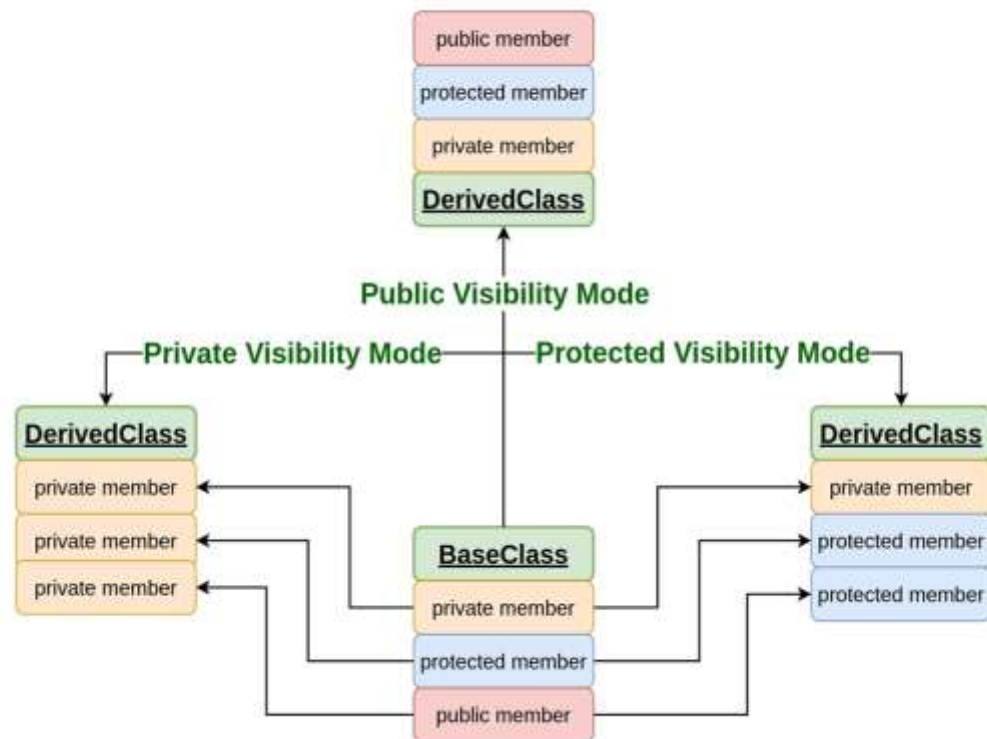
Public mode: If we derive a sub class from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in derived class.

Protected mode: If we derive a sub class from a Protected base class. Then both public member and protected members of the base class will become protected in derived class.

Private mode: If we derive a sub class from a Private base class. Then both public member and protected members of the base class will become Private in derived class.



Visibility Modes in C++



Summarized Accessibility:

Base class member access specifier	Type of Inheritance		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not accessible (Hidden)	Not accessible (Hidden)	Not accessible (Hidden)

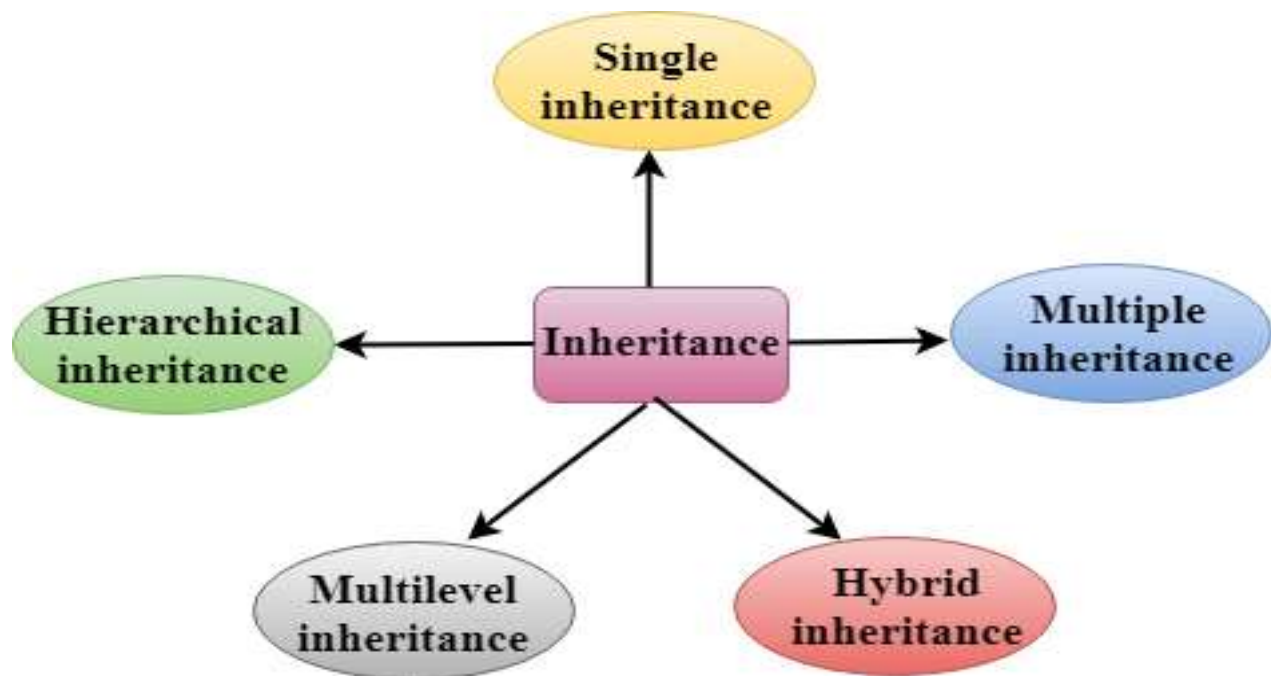
```
class A{  
public:  
    int x;  
protected:  
    int y;  
private:  
    int z;  
};  
  
class B : public A  
{  
    // x is public  
    // y is protected  
    // z is not accessible from B  
};
```

```
class C : protected A  
{  
    // x is protected  
    // y is protected  
    // z is not accessible from C  
};
```

```
class D : private A // 'private' is default for classes  
{  
    // x is private  
    // y is private  
    // z is not accessible from D  
};
```

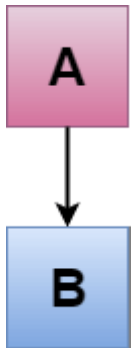
C++ supports five types of inheritance:

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



1. Single Inheritance:

In single inheritance, a class is allowed to inherit from only one class. i.e. one sub class is inherited by one base class only.



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

Example:

```
#include <iostream>

using namespace std;

class A //base class
{
    int a = 4;
    int b = 5;
    public:
    int mul()
    {
        int c = a*b;
```

```
        return c;
    }
};

class B : private A //child class
{
    public:
    void display()
    {
        int result = mul();

        std::cout <<"Multiplication of a and b is : "<<result<<
std::endl;
    }
};

int main()
{
    B b;

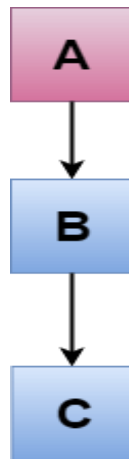
    b.display();

    return 0;
}
```

```
Multiplication of a and b is : 20

...Program finished with exit code 0
Press ENTER to exit console.□
```

2. Multilevel Inheritance:



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

When one class inherits another class which is further inherited by another class, it is known as multilevel inheritance in C++.

Inheritance is transitive so the last derived class acquires all the members of all its base classes.

Syntax :

```
class A // base class
```

```
{
```

```

.....
};
class B : access_specifier A // derived class
{
    .....
} ;
class C : access_specifier B // derived from derived class B
{
    .....
} ;

```

Example:

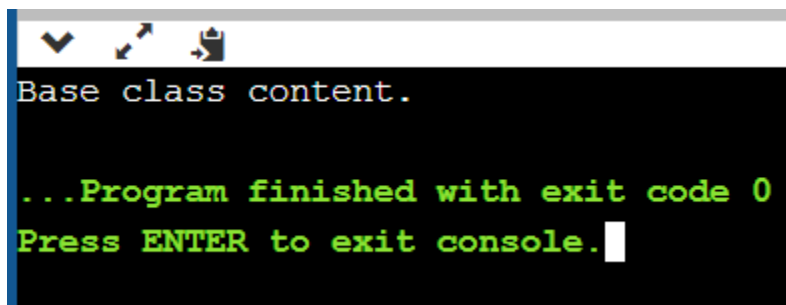
```

#include <iostream>
using namespace std;
class A
{
    public:
        void display()
        {
            cout<<"Base class content.";
        }
}

```

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

```
int main()  
{  
    C obj;  
    obj.display();  
    return 0;  
}
```

A screenshot of a console window with a black background and white and green text. The window has a title bar with standard icons. The text displayed is: "Base class content." in white, followed by "...Program finished with exit code 0" and "Press ENTER to exit console." in green. A white cursor is visible at the end of the last line.

```
Base class content.  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Example 2:

```
#include <iostream>

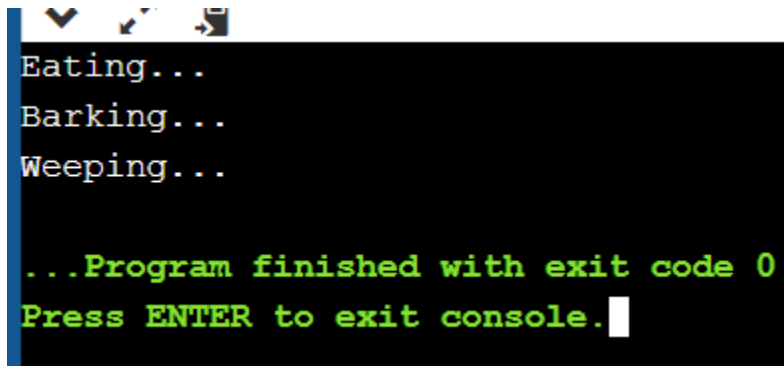
using namespace std;

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() {
```

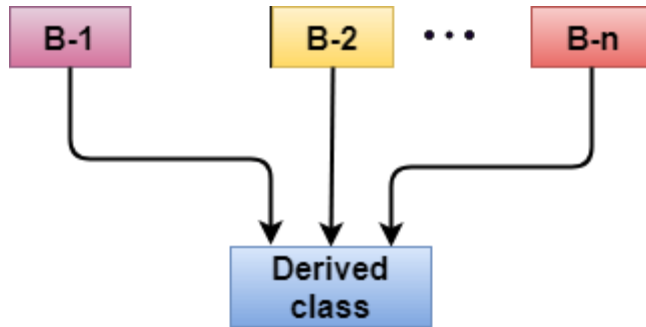
```
        cout<<"Weeping...";  
    }  
};  
  
int main(void) {  
    BabyDog d1;  
    d1.eat();  
    d1.bark();  
    d1.weep();  
    return 0;  
}
```

A terminal window with a black background and a blue title bar. The title bar contains three icons: a downward arrow, a magnifying glass, and a document. The terminal displays the output of the program: "Eating...", "Barking...", and "Weeping...". Below these, it shows "...Program finished with exit code 0" and "Press ENTER to exit console." with a white cursor at the end.

```
Eating...  
Barking...  
Weeping...  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

3. Multiple Inheritance

Multiple Inheritance is a feature of C++ where a class can inherit from more than one classes. i.e one sub class is inherited from more than one base classes.



Syntax:

```
Class child: public base_class1, protected base_class2, ....
```

```
{
```

```
    //body of subclass
```

```
};
```

Here, the number of base classes will be separated by a comma (‘, ‘) and access mode for every base class must be specified

Example:

```
#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()

{

cout << "The value of a is : " <<a<< endl;

cout << "The value of b is : " <<b<< endl;

cout<<"Addition of a and b is : "<<a+b;

}

};

int main()

{

C c;

c.get_a(10);

c.get_b(20);

c.display();

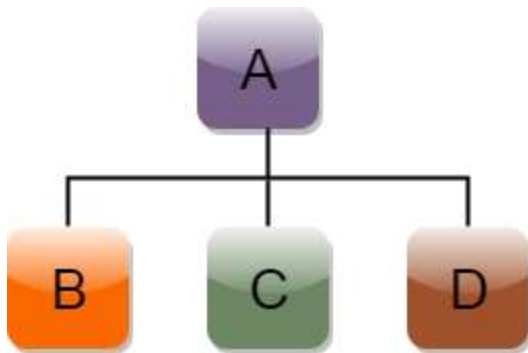
return 0;

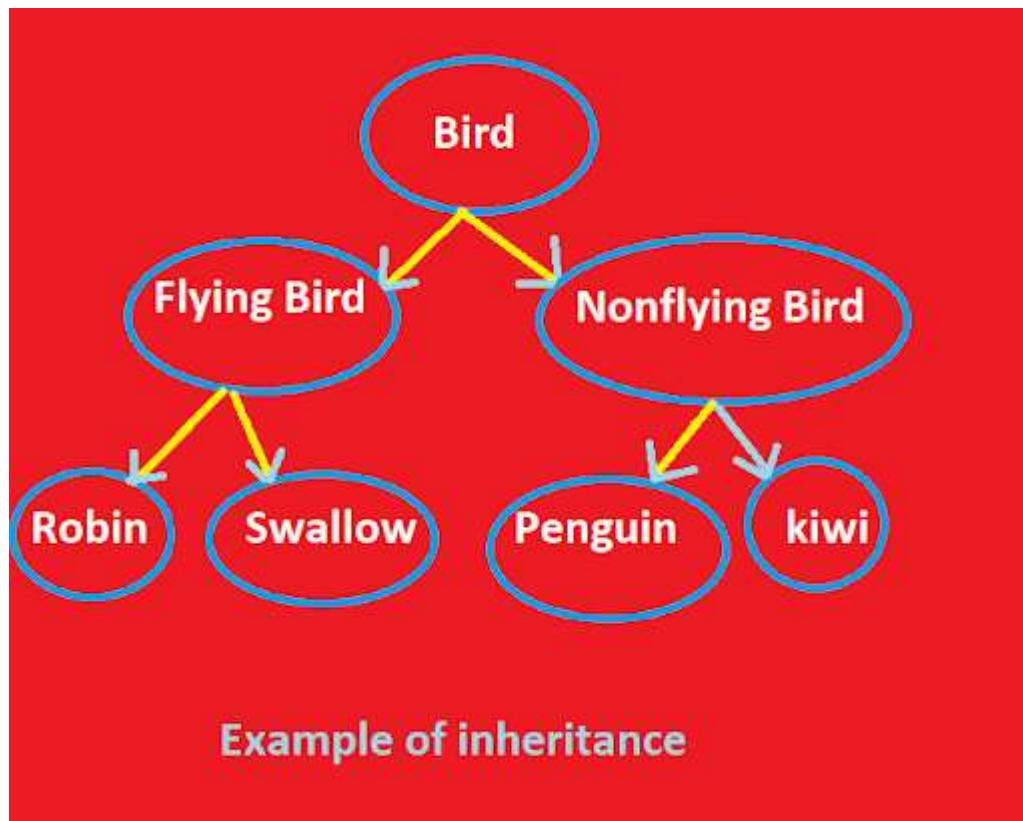
}

```
The value of a is : 10  
The value of b is : 20  
Addition of a and b is : 30  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

4. Hierarchical Inheritance:

In this type of inheritance, more than one sub class is inherited from a single base class. i.e. more than one derived class is created from a single 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.  
}
```

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()
```

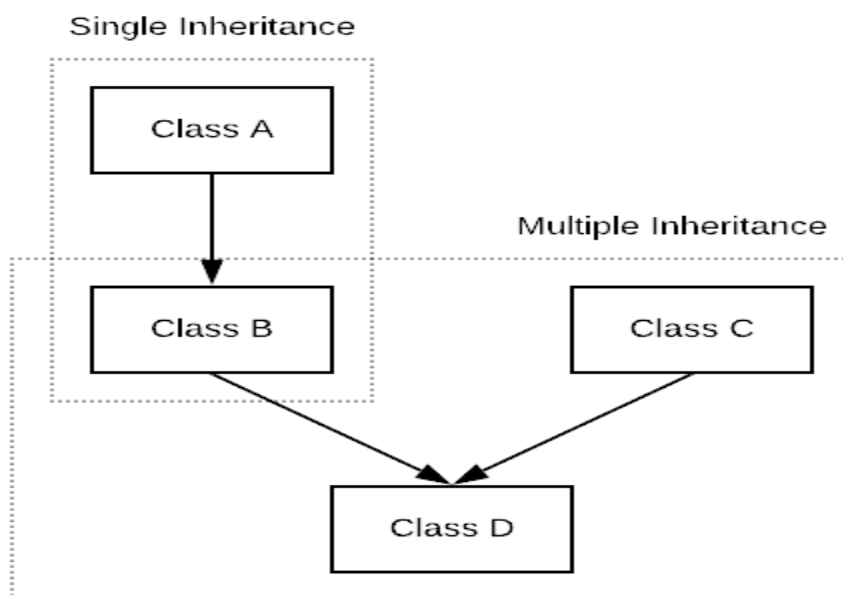
```
{  
    Rectangle r;  
    Triangle t;  
    int length,breadth,base,height;  
    std::cout << "Enter the length and breadth of a rectangle: " <<  
std::endl;  
    cin>>length>>breadth;  
    r.get_data(length,breadth);  
    int m = r.rect_area();  
    std::cout << "Area of the rectangle is :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

```
Enter the length and breadth of a rectangle:
6
8
Area of the rectangle is : 48
Enter the base and height of the triangle:
7
9
Area of the triangle is : 31

...Program finished with exit code 0
Press ENTER to exit console.
```

5. Hybrid Inheritance:

Hybrid Inheritance is implemented by combining more than one type of inheritance.



Syntax for above diagram:

Class A

{

};

Class B:public A

{

//codes

};

Class C:public A

{

};

Class D: public B, public C

{

};

Example:1

```
#include<iostream>
```

```
using namespace std;
```

```
int a,b,c,d,e;
```

```
class A
```

```
{  
protected:  
public:  
    void getab()  
    {  
        cout<<"\nEnter a and b value:";  
        cin>>a>>b;  
    }  
};
```

```
class B:public A {  
protected:  
public:  
    void getc()  
    {  
        cout<<"Enter c value:";  
        cin>>c;  
    }  
};
```

```
class C
{
protected:
public:
    void getd()
    {
        cout<<"Enter d value:";
        cin>>d;
    }
};
```

```
class D:public B,public C
{
protected:
public:
    void result()
    {
        getab();
        getc();
        getd();
    }
};
```

```
        e=a+b+c+d;

        cout<<"\n Addition is :"<<e;

    }

};

int main()

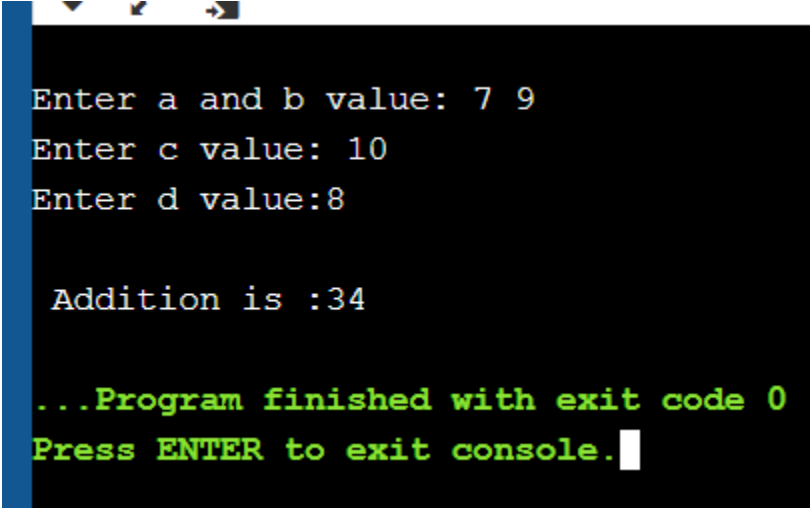
{

    D d1;

    d1.result();


    return 0;

}
```



```
Enter a and b value: 7 9
```

```
Enter c value: 10
```

```
Enter d value:8
```

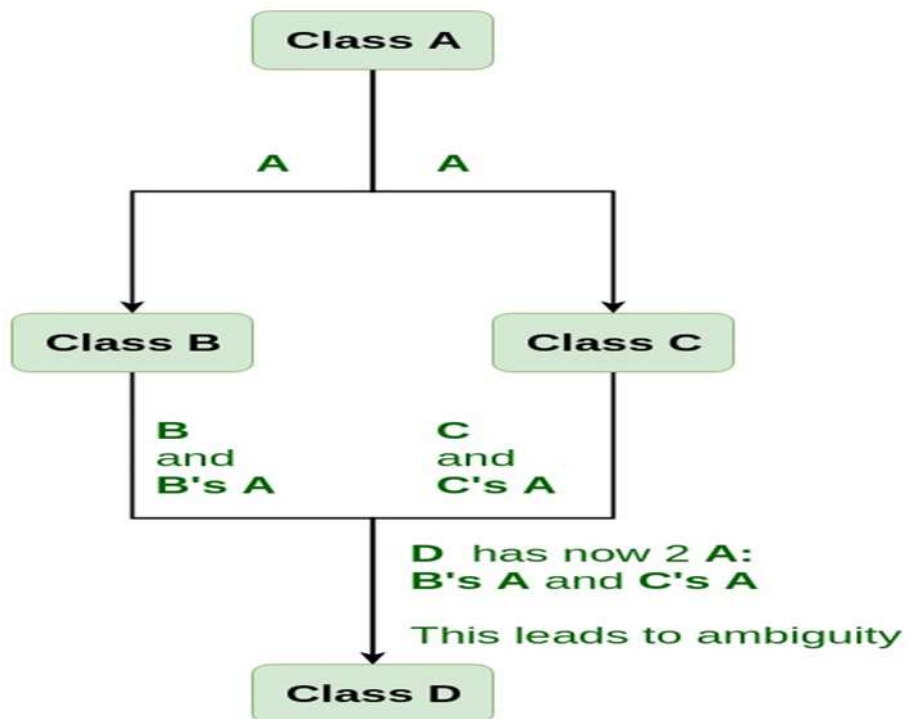
```
    Addition is :34
```

```
...Program finished with exit code 0
```

```
Press ENTER to exit console.
```

*****Ambiguity in Hybrid Inheritance when as a type multiple inheritance (Multipath inheritance) is being used****

See the below diagram:



```
class A {  
    public:  
        int a;  
};  
  
class B : public A {  
    public:  
        int b;  
};  
  
class C : public A {  
    public:  
        int c;  
};  
  
class D : public B, public C {  
    public:  
        int d;  
};
```

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  
{  
  
};  
  
class D:public B,public C  
{  
    void view()  
    {  
        display();  
    }  
};
```

```
int main()
{
    C d;

    d.display();

    return 0;
}
```

```
main.cpp: In member function 'void D::view()':
main.cpp:27:9: error: reference to 'display' is ambiguous
    display();
    ^~~~~~
main.cpp:6:10: note: candidates are: void A::display()
    void display()
    ^~~~~~
main.cpp:14:10: note:                  void B::display()
    void display()
    ^~~~~~
```

How to resolve these kind of Ambiguities????

Solution is “Virtual Base Class”

What is Virtual Base class?

- An ambiguity can arise when several paths exist to a class from the same base class. This means that a child class could have duplicate sets of members inherited from a single base class.
- C++ solves this issue by introducing a virtual base class. When a class is made virtual, necessary care is taken so that the duplication is avoided regardless of the number of paths that exist to the child class.

This can be achieved by preceding the base class' name with the word **virtual**.

Solution for above example

```
class A {  
    public:  
        int a;  
};  
  
class B: virtual public A{    //virtual inheritance  
    public:  
        int b;  
};  
  
class C : virtual public A {    //virtual inheritance  
    public:  
        int c;  
};  
  
class D: public B, public C {  
    public:  
        int d;  
};
```

Example:

```
#include <iostream>
```

```
using namespace std;
```

```
class A
```

```
{
```

```
    public:
```

```
        int i;
```

```
};
```

```
class B : virtual public A
```

```
{
```

```
    public:
```

```
        int j;
```

```
};
```

```
class C: virtual public A
```

```
{
```

```
    public:
```

```
        int k;
```

```
};
```

```
class D: public B, public C
```

```
{
```

```
    public:
```

```
        int sum;
```

```
};
```

```
int main()
```

```
{
```

```
    D ob;
```

```
    ob.i = 10; //unambiguous since only one copy of i is inherited.
```

```
    ob.j = 20;
```

```
    ob.k = 30;
```

```
    ob.sum = ob.i + ob.j + ob.k;
```

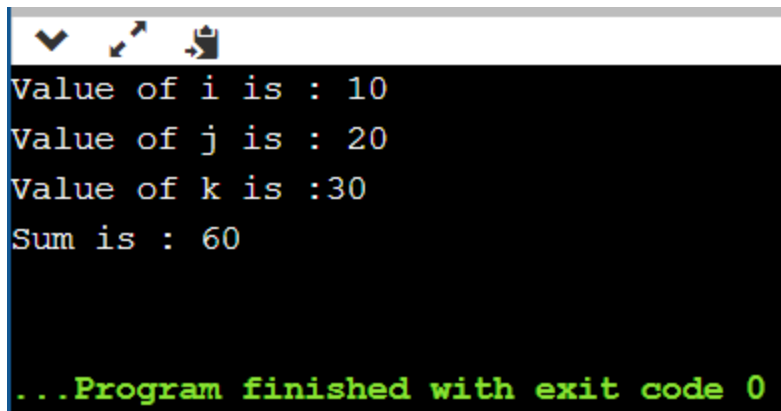
```
    cout << "Value of i is : " << ob.i << endl;
```

```
    cout << "Value of j is : " << ob.j << "\n"; cout << "Value of k is  
:" << ob.k << endl;
```

```
    cout << "Sum is : " << ob.sum << endl;
```

```
    return 0;
```

```
}
```

A screenshot of a terminal window with a black background and white text. The text shows the output of a C++ program: 'Value of i is : 10', 'Value of j is : 20', 'Value of k is :30', and 'Sum is : 60'. At the bottom, in green text, it says '...Program finished with exit code 0'. The terminal window has a standard macOS-style title bar with a red close button, a yellow maximize button, and a green full-screen button.

```
Value of i is : 10
Value of j is : 20
Value of k is :30
Sum is : 60

...Program finished with exit code 0
```

Function overriding

As we know, inheritance is a feature of OOP that allows us to create derived classes from a base class. The derived classes inherit features of the base class.

Suppose, the same function is defined in both the derived class and the based class. Now if we call this function using the object of the derived class, the function of the derived class is executed.

This is known as function overriding in C++. The function in derived class overrides the function in base class.

Example:

```
// C++ program to demonstrate function overriding
```

```
#include <iostream>
```

```
using namespace std;
```

```
class Base {
```

```
public:
    void print() {
        cout << "Base Function" << endl;
    }
};

class Derived : public Base {
public:
    void print() {
        cout << "Derived Function" << endl;
    }
};

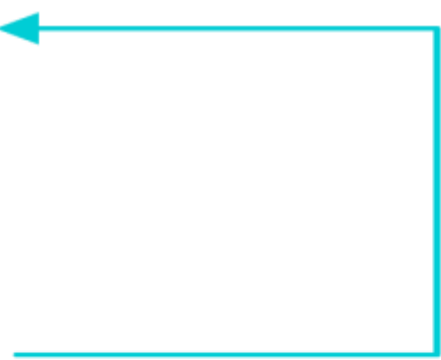
int main() {
    Derived derived1;
    derived1.print();
    return 0;
}
```

Output.....

Derived Function

How it worked logically?

```
class Base {  
    public:  
        void print() {  
            // code  
        }  
};  
  
class Derived : public Base {  
    public:  
        void print() {  
            // code  
        }  
};  
  
int main() {  
    Derived derived1;  
    derived1.print();  
    return 0;  
}
```

A blue arrow originates from the `derived1.print();` line in the `main()` function and points to the `void print() {` line within the `Derived` class definition, illustrating how the compiler resolves the function call to the derived class's implementation.

How to resolve Function overriding or how to Access overridden function?

To access the overridden function of the base class, we use the **scope resolution operator ::**.

We can also access the overridden function by using a pointer of the base class to point to an object of the derived class and then calling the function from that pointer.

// C++ program to access overridden function

// in main() using the scope resolution operator ::

```
#include <iostream>
```

```
using namespace std;
```

```
class Base {
```

```
    public:
```

```
    void print() {
```

```
        cout << "Base Function" << endl;
```

```
    }
```

```
};
```

```
class Derived : public Base {
```

```
    public:
```

```
    void print() {
```

```
        cout << "Derived Function" << endl;
```

```
    }
```

```
};
```

```
int main() {
```

```
    Derived derived1, derived2;
```

```
    derived1.print();
```

// access print() function of the Base class

derived2.Base::print();

return 0;

}

Output....

Derived Function

Base Function

```
class Base {  
    public:  
    void print() {  
        // code  
    }  
};  
  
class Derived : public Base {  
    public:  
    void print() {  
        // code  
    }  
};  
  
int main() {  
    Derived derived1, derived2;  
  
    derived1.print();  
  
    derived2.Base::print();  
  
    return 0;  
}
```

The diagram illustrates the function calls made in the main() function. A horizontal line from `derived1.print();` leads to the `print()` method of the `Derived` class. A horizontal line from `derived2.Base::print();` leads to the `print()` method of the `Base` class. Both horizontal lines are part of a larger cyan structure that includes vertical lines extending downwards and then turning back to the left to point at the respective `print()` methods.