<u>Inheritance:</u>

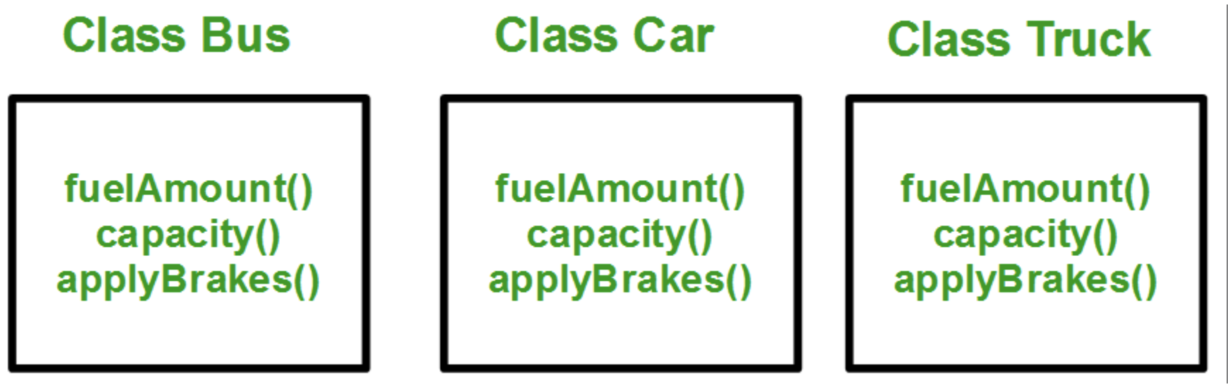
Inheritance is a feature or a process in which, new classes are created from the existing classes. The new class created is called “derived class” or “child class” and the existing class is known as the “base class” or “parent class”. The derived class now is said to be inherited from the base class.

When we say derived class inherits the base class, it means, the derived class inherits all the properties of the base class, without changing the properties of base class and may add new features to its own. These new features in the derived class will not affect the base class. The derived class is the specialized class for the base class.

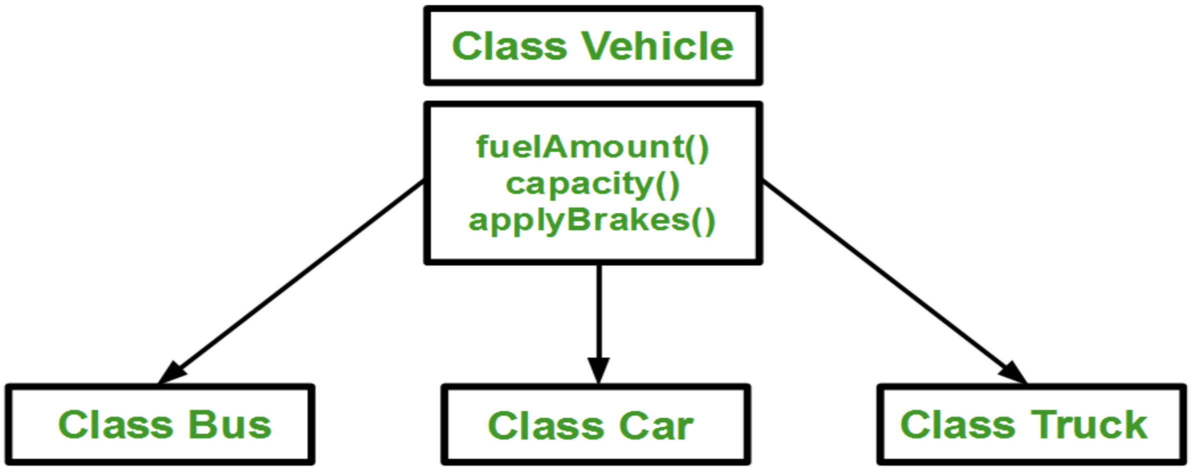
- **Sub Class:** The class that inherits properties from another class is called Subclass or Derived Class.
- **Super Class:** The class whose properties are inherited by a subclass is called Base Class or Superclass.

Why and when to use inheritance?

Consider a group of vehicles. You need to create classes for Bus, Car, and Truck. The methods fuelAmount(), capacity(), applyBrakes() will be the same for all three classes. If we create these classes avoiding inheritance then we have to write all of these functions in each of the three classes as shown below figure:



You can clearly see that the above process results in duplication of the same code 3 times. This increases the chances of error and data redundancy. To avoid this type of situation, inheritance is used. If we create a class Vehicle and write these three functions in it and inherit the rest of the classes from the vehicle class, then we can simply avoid the duplication of data and increase re-usability. Look at the below diagram in which the three classes are inherited from vehicle class:



Using inheritance, we have to write the functions only one time instead of three times as we have inherited the rest of the three classes from the base class (Vehicle).

Implementing inheritance in C++: For creating a sub-class that is inherited from the base class we have to follow the below syntax.

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

Syntax:

```
class  <derived_class_name> : <access-specifier> <base_class_name>
{
      //body
}
```

Where
class — keyword to create a new class
derived_class_name — name of the new class, which will inherit the base class
access-specifier — either of private, public or protected. If neither is specified, PRIVATE is taken as default
base-class-name — name of the base class

Example:

```
1. class ABC : private XYZ          //private derivation
         {            }
2. class ABC : public XYZ          //public derivation
         {           }
3. class ABC : protected XYZ          //protected derivation
         {          }
4. class ABC: XYZ                    //private derivation by default
{         }
```

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.

is-a relationship

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

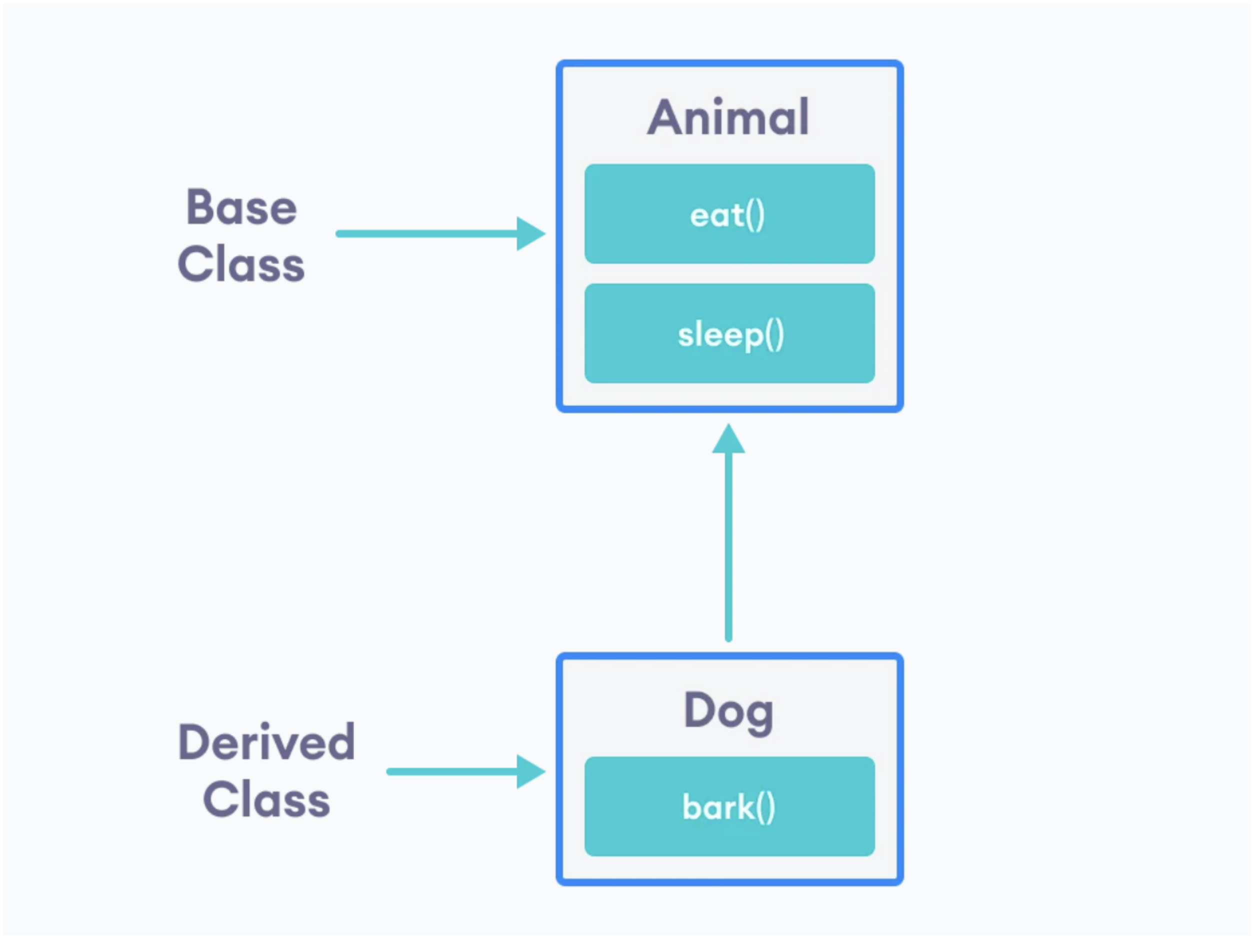
Here are some examples:

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

The derived class inherits the features from the base class and can have additional features of its own. For example,

```
class Animal {
   // eat() function
   // sleep() function
};

class Dog : public Animal {
   // bark() function
};
```

Here, the Dog class is derived from the Animal class. Since Dog is derived from Animal, members of Animal are accessible to Dog.



Example 1: Simple Example of C++ Inheritance

```
// C++ program to demonstrate inheritance

#include <iostream>
using namespace std;

// base class
class Animal {

  public:
        void eat() {
        cout << "I can eat!" << endl;
        }

        void sleep() {
        cout << "I can sleep!" << endl;
        }
};

// derived class
class Dog : public Animal {

  public:
        void bark() {
        cout << "I can bark! Woof woof!!" << endl;
        }
};

int main() {
        // Create object of the Dog class
        Dog dog1;

        // Calling members of the base class
        dog1.eat();
        dog1.sleep();

        // Calling member of the derived class
        dog1.bark();

        return 0;
}
```

Output

```
I can eat!
I can sleep!
I can bark! Woof woof!!
```

Here, dog1 (the object of derived class Dog) can access members of the base class Animal. It's because Dog is inherited from Animal.

C++ protected Members

The access modifier protected is especially relevant when it comes to C++ inheritance.

Like private members, protected members are inaccessible outside of the class. However, they can be accessed by **derived classes** and **friend classes/functions**.

We need protected members if we want to hide the data of a class, but still want that data to be inherited by its derived classes.

Example 2 : C++ protected Members

```
// C++ program to demonstrate protected members

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

// base class
class Animal {

  private:
        string color;

  protected:
        string type;

  public:
        void eat() {
        cout << "I can eat!" << endl;
        }

        void sleep() {
        cout << "I can sleep!" << endl;
        }

        void setColor(string clr) {
        color = clr;
        }

        string getColor() {
        return color;
        }
};

// derived class
class Dog : public Animal {

  public:
        void setType(string tp) {
        type = tp;
        }

        void displayInfo(string c) {
        cout << "I am a " << type << endl;
        cout << "My color is " << c << endl;
        }

        void bark() {
        cout << "I can bark! Woof woof!!" << endl;
        }
};

int main() {
        // Create object of the Dog class
        Dog dog1;

        // Calling members of the base class
        dog1.eat();
        dog1.sleep();
        dog1.setColor("black");

        // Calling member of the derived class
        dog1.bark();
        dog1.setType("mammal");

        // Using getColor() of dog1 as argument
        // getColor() returns string data
        dog1.displayInfo(dog1.getColor());

        return 0;
}
```

Output

```
I can eat!
I can sleep!
I can bark! Woof woof!!
I am a mammal
My color is black
```

Here, the variable type is protected and is thus accessible from the derived class Dog. We can see this as we have initialized type in the Dog class using the function setType().

On the other hand, the private variable color cannot be initialized in Dog.

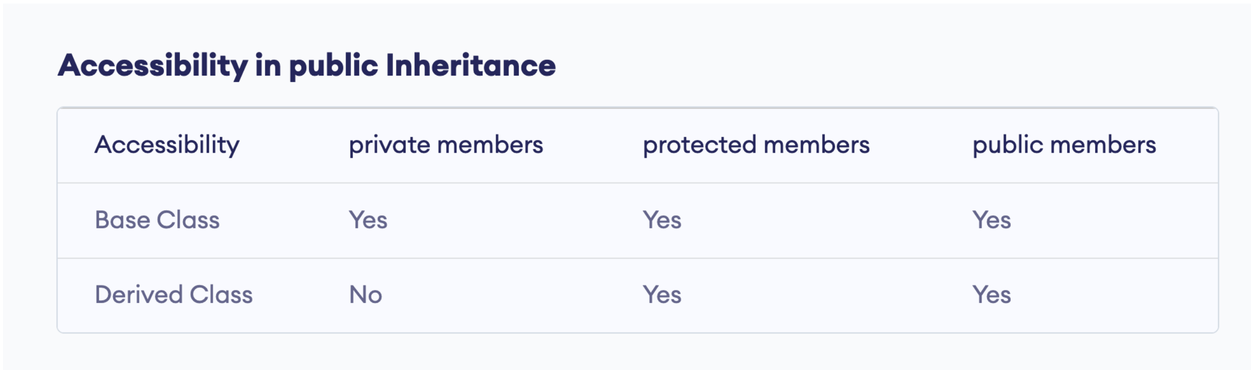
public, protected and private inheritance in C++

public, **protected,** and **private** inheritance have the following features:

- **public inheritance** makes public members of the base class public in the derived class, and the protected members of the base class remain protected in the derived class.
- **protected inheritance** makes the public and protected members of the base class protected in the derived class.
- **private inheritance** makes the public and protected members of the base class private in the derived class.

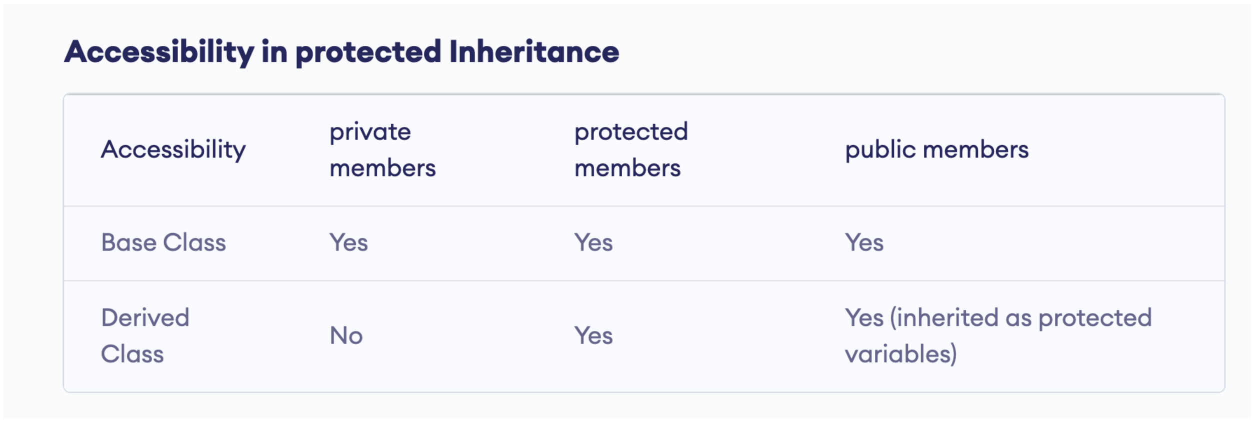
Note: private members of the base class are inaccessible to the derived class.

```
class Base {
  public:
    int x;
  protected:
    int y;
  private:
    int z;
};

class PublicDerived: public Base {
  // x is public
  // y is protected
  // z is not accessible from PublicDerived
};

class ProtectedDerived: protected Base {
  // x is protected
  // y is protected
  // z is not accessible from ProtectedDerived
};

class PrivateDerived: private Base {
  // x is private
  // y is private
  // z is not accessible from PrivateDerived
};
```



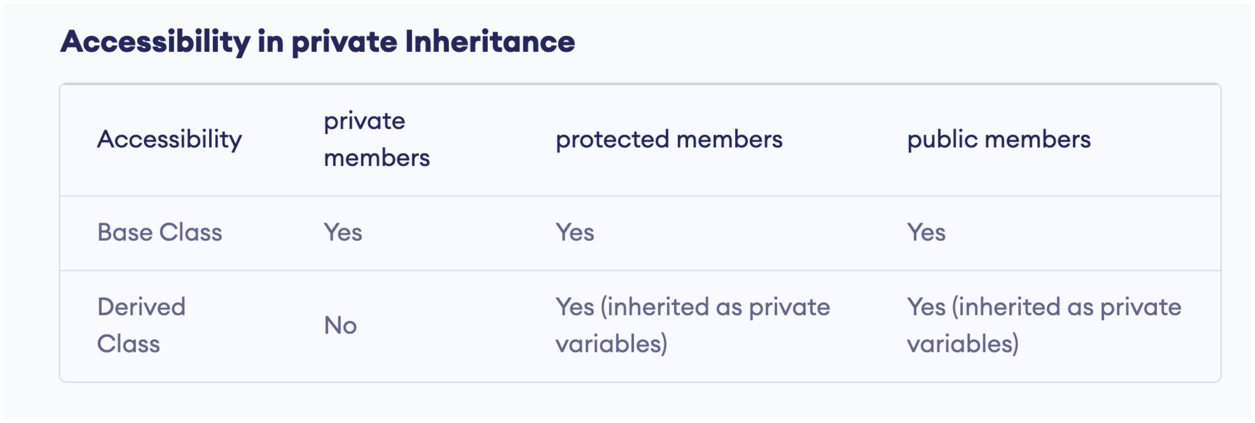
Accessibility in public Inheritance

Accessibility	private members	protected members	public members
Base Class	Yes	Yes	Yes
Derived Class	No	Yes	Yes



Accessibility in protected Inheritance

Accessibility	private members	protected members	public members
Base Class	Yes	Yes	Yes
Derived Class	No	Yes	Yes (inherited as protected variables)



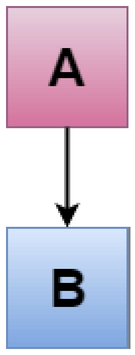
Accessibility in private Inheritance

Accessibility	private members	protected members	public members
Base Class	Yes	Yes	Yes
Derived Class	No	Yes (inherited as private variables)	Yes (inherited as private variables)

Types Of Inheritance:-

1. Single inheritance
2. Multilevel inheritance
3. Multiple inheritance
4. Hierarchical inheritance
5. Hybrid inheritance

1. Single Inheritance: In single inheritance, a class is allowed to inherit from only one class. i.e. one subclass is inherited by one base class only.



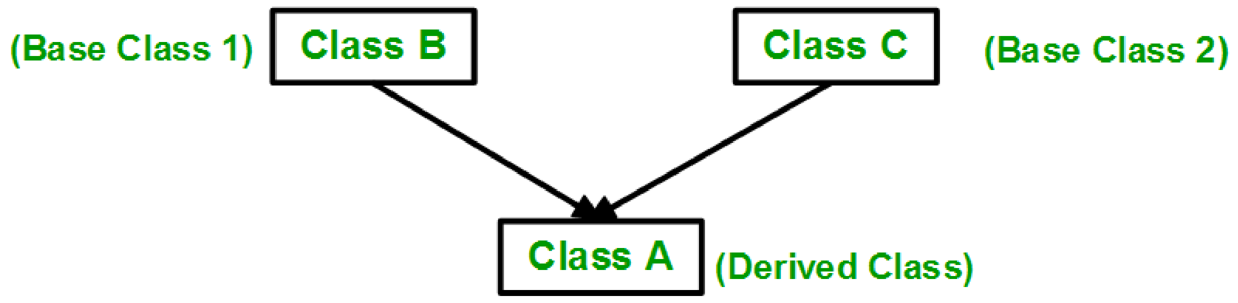
Where 'A' is the base class, and 'B' is the derived class.

```
// Example:

#include<iostream>
using namespace std;

class A
{
   protected:
   int a;

   public:
         void set_A()
         {
                  cout<<"Enter the Value of A=";
                  cin>>a;

         }
         void disp_A()
         {
                  cout<<endl<<"Value of A="<<a;
         }
};


class B: public A
{
   int b,p;

   public:
         void set_B()
         {
                  set_A();
                  cout<<"Enter the Value of B=";
```

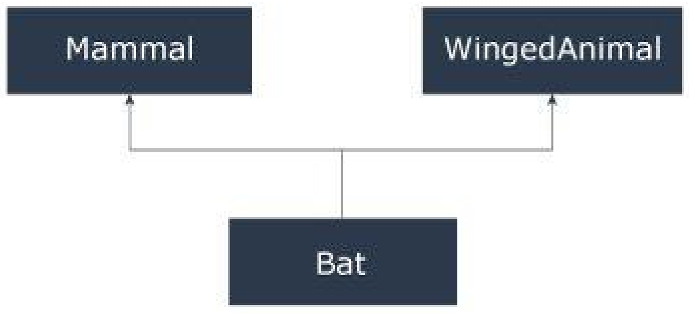
```
                        cin>>b;
                }

                void disp_B()
                {
                        disp_A();
                        cout<<endl<<"Value of B="<<b;
                }

                void cal_product()
                {
                        p=a*b;
                        cout<<endl<<"Product of "<<a<<" * "<<b<<" = "<<p;
                }

};

main()
{

   B _b;
   _b.set_B();
   _b.cal_product();

   return 0;

}


// Example: 2

#include<iostream>
using namespace std;

class A
{
   protected:
   int a;

   public:
         void set_A()
         {
                 cout<<"Enter the Value of A=";
                 cin>>a;

         }
         void disp_A()
         {
                 cout<<endl<<"Value of A="<<a;
         }
};
```

```
class B: public A
{
   int b,p;

   public:
          void set_B()
          {
                 set_A();
                 cout<<"Enter the Value of B=";
                 cin>>b;
          }

          void disp_B()
          {
                 disp_A();
                 cout<<endl<<"Value of B="<<b;
          }

          void cal_product()
          {
                 p=a*b;
                 cout<<endl<<"Product of "<<a<<" * "<<b<<" = "<<p;
          }

};

main()
{

   B _b;
   _b.set_B();
   _b.cal_product();

   return 0;

}
```

2. Multiple Inheritance: Multiple Inheritance is a feature of C++ where a class can inherit from more than one class. i.e one **subclass** is inherited from more than one **base class**.



In C++ programming, a class can be derived from more than one parent. For example, A class Bat is derived from base classes Mammal and WingedAnimal. It makes sense because bat is a mammal as well as a winged animal.



Multiple Inheritance

Syntax:

```
class subclass_name : access_mode base_class1, access_mode base_class2, ....
{
  // body of subclass
};


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

```
{
... ... ...
};
```

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

class Mammal {
  public:
    Mammal() {
      cout << "Mammals can give direct birth." << endl;
    }
};

class WingedAnimal {
  public:
    WingedAnimal() {
      cout << "Winged animal can flap." << endl;
    }
};

class Bat: public Mammal, public WingedAnimal {};

int main() {
    Bat b1;
    return 0;
}
```

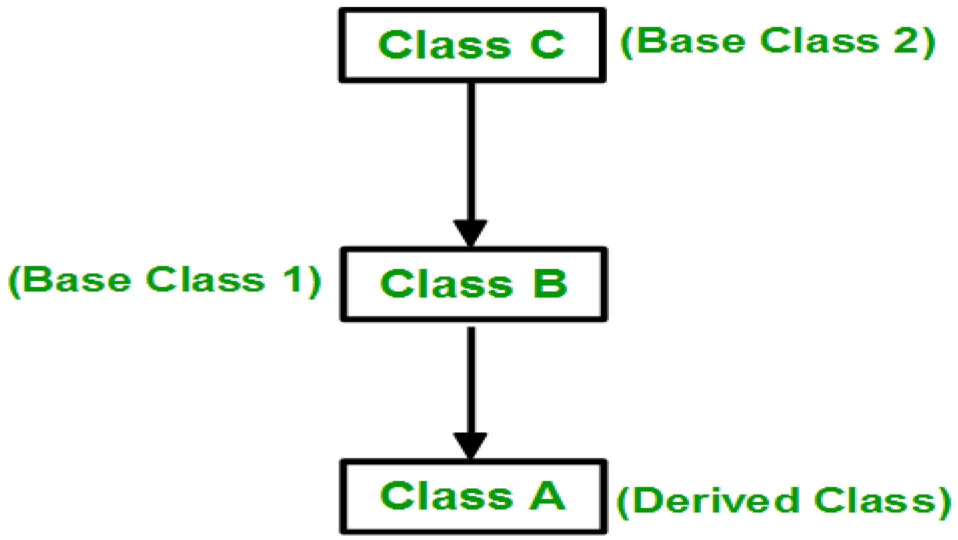
Output

```
Mammals can give direct birth.
Winged animal can flap.
```

```
 // C++ program to explain
// multiple inheritance
#include <iostream>
using namespace std;

// first base class
class Vehicle {
public:
    Vehicle() { cout << "This is a Vehicle\n"; }
};
```

```
// second base class
class FourWheeler {
public:
    FourWheeler()
    {
            cout << "This is a 4 wheeler Vehicle\n";
    }
};

// sub class derived from two base classes
class Car : public Vehicle, public FourWheeler {
};

// main function
int main()
{
    // Creating object of sub class will
    // invoke the constructor of base classes.
    Car obj;
    return 0;
}
```

3. Multilevel Inheritance: In this type of inheritance, a derived class is created from another derived class.

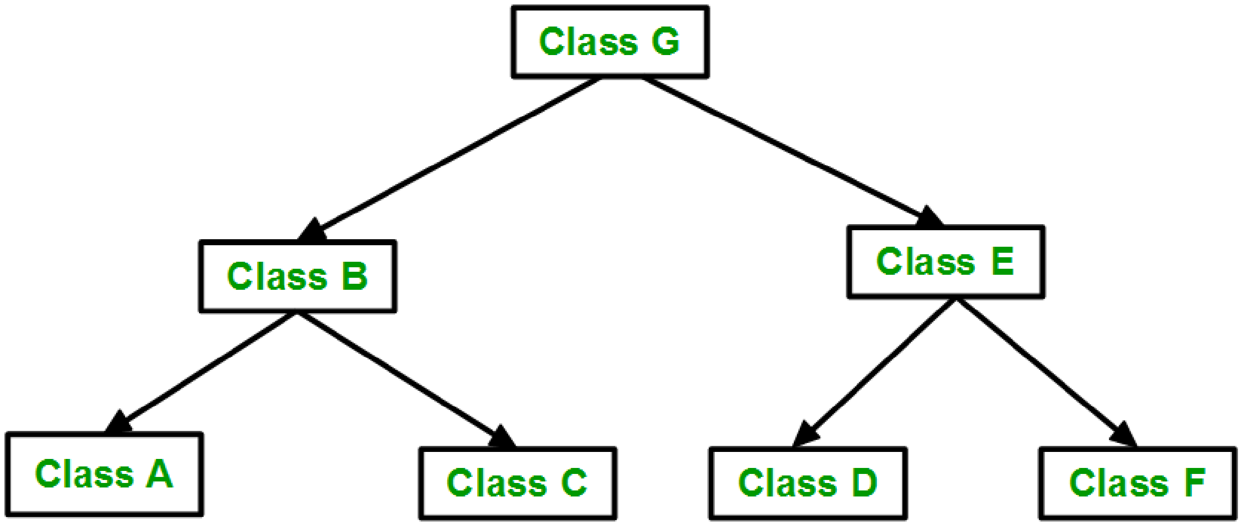


Syntax:-

class C

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

#include <iostream>
using namespace std;

// base class
class Vehicle {
public:
    Vehicle() { cout << "This is a Vehicle\n"; }
};

// first sub_class derived from class vehicle
class fourWheeler : public Vehicle {
public:
    fourWheeler()
    {
        cout << "Objects with 4 wheels are vehicles\n";
    }
};
// sub class derived from the derived base class fourWheeler
class Car : public fourWheeler {
public:
    Car() { cout << "Car has 4 Wheels\n"; }
};

// main function
int main()
{
    // Creating object of sub class will
    // invoke the constructor of base classes.
    Car obj;
    return 0;
}
```

4. Hierarchical Inheritance: In this type of inheritance, more than one subclass is inherited from a single base class. i.e. more than one derived class is created from a single base class.



Syntax:-

```
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.
}


// C++ program to demonstrate hierarchical inheritance
#include <iostream>
using namespace std;

// base class
class Animal {
   public:
    void info() {
        cout << "I am an animal." << endl;
    }
};

// derived class 1
class Dog : public Animal {
   public:
    void bark() {
        cout << "I am a Dog. Woof woof." << endl;
    }
};

// derived class 2
class Cat : public Animal {
   public:
    void meow() {
        cout << "I am a Cat. Meow." << endl;
    }
};

int main() {
    // Create object of Dog class
    Dog dog1;
    cout << "Dog Class:" << endl;
    dog1.info();  // Parent Class function
    dog1.bark();

    // Create object of Cat class
    Cat cat1;
    cout << "\nCat Class:" << endl;
    cat1.info();  // Parent Class function
    cat1.meow();

    return 0;
}
```

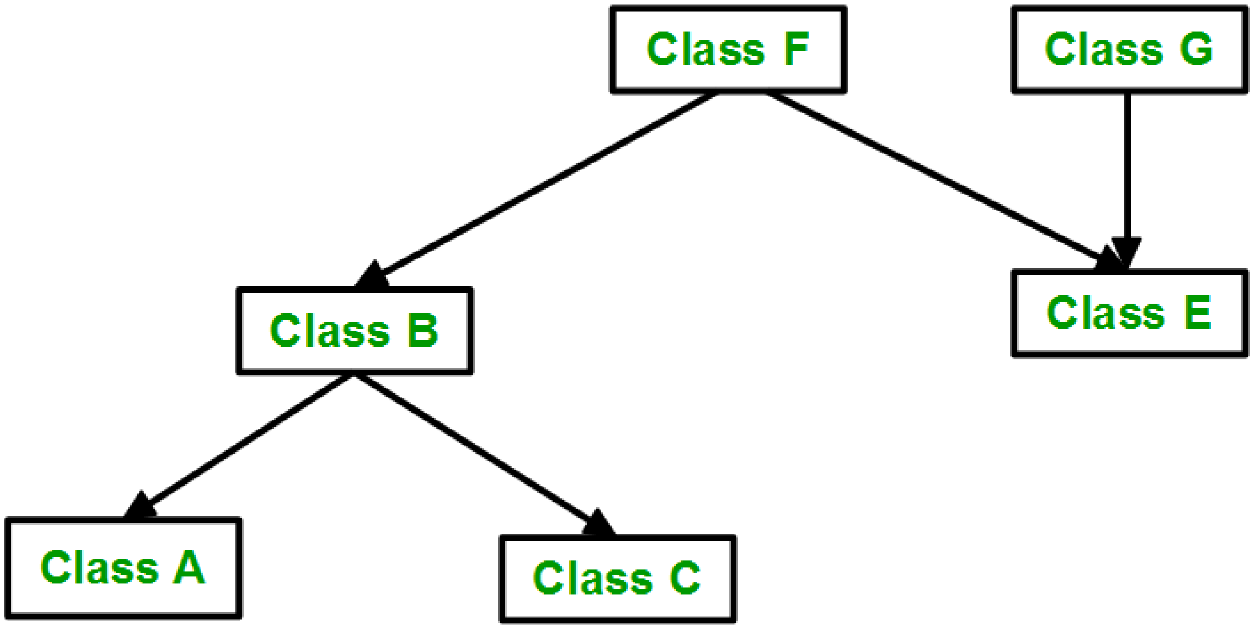
Here, both the Dog and Cat classes are derived from the Animal class. As such, both the derived classes can access the info() function belonging to the Animal class.

```
// C++ program to implement
// Hierarchical Inheritance
#include <iostream>
```

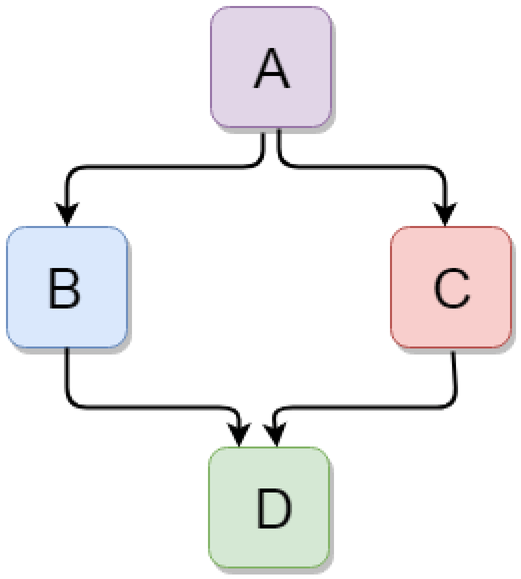
```
using namespace std;

// base class
class Vehicle {
public:
    Vehicle() { cout << "This is a Vehicle\n"; }
};

// first sub class
class Car : public Vehicle {
};

// second sub class
class Bus : public Vehicle {
};

// main function
int main()
{
    // Creating object of sub class will
    // invoke the constructor of base class.
    Car obj1;
    Bus obj2;
    return 0;
}
```

5. Hybrid (Virtual) Inheritance: Hybrid Inheritance is implemented by combining more than one type of inheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance.

Below image shows the combination of hierarchical and multiple inheritances:



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++ program for Hybrid Inheritance

#include <iostream>
using namespace std;

// base class
class Vehicle {
public:
   Vehicle() { cout << "This is a Vehicle\n"; }
};

// base class
class Fare {
public:
   Fare() { cout << "Fare of Vehicle\n"; }
};

// first sub class
class Car : public Vehicle {
};

// second sub class
class Bus : public Vehicle, public Fare {
};

// main function
int main()
{
   // Creating object of sub class will
```

```
    // invoke the constructor of base class.
    Bus obj2;
    return 0;
}
```

Output

```
This is a Vehicle
Fare of Vehicle
```

```
// Example:

#include <iostream>
using namespace std;

class A
{
    protected:
    int a;
    public:
    void get_a()
    {
    cout << "Enter the value of 'a' : ";
    cin>>a;
    }
};

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

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


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

Output:

```
Enter the value of 'a' : 12
Enter the value of 'b' : 34
Enter the value of c is : 5
Multiplication of a,b,c is : 2040
```