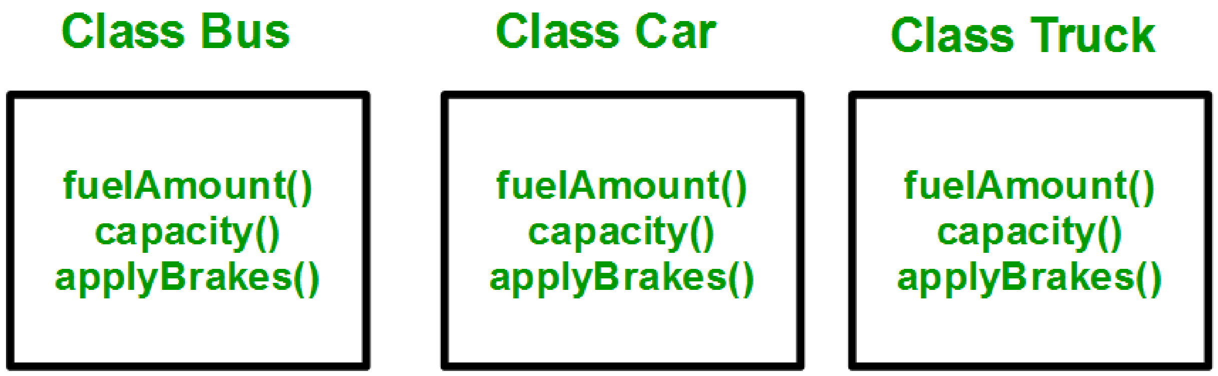
Inheritance in C++

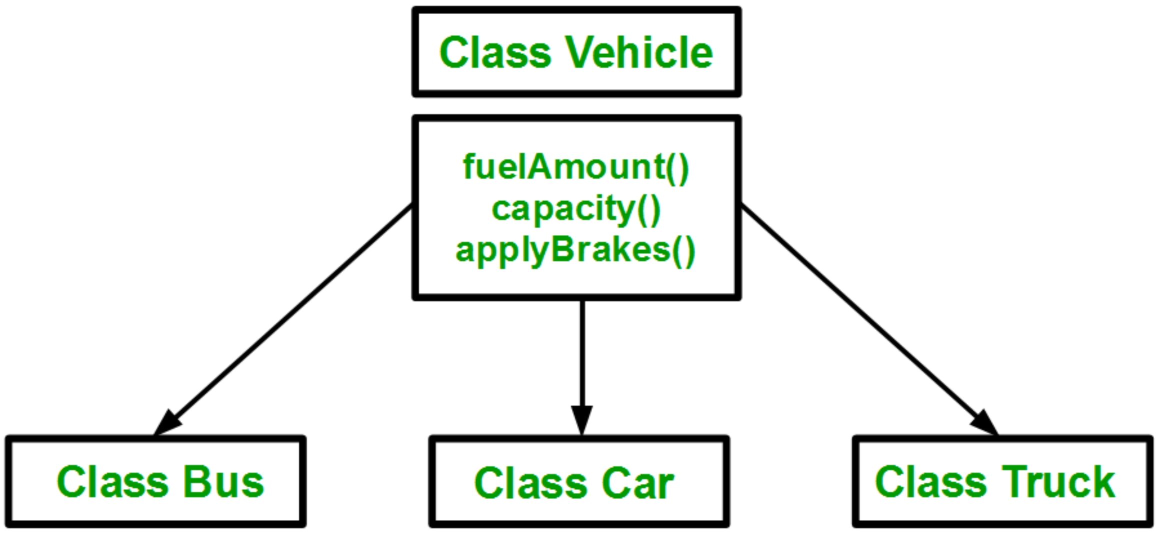
The capability of a class to derive properties and characteristics from another class is called **Inheritance**. Inheritance is one of the most important feature of Object Oriented Programming.
Sub Class: The class that inherits properties from another class is called Sub class or Derived Class.

Consider a group of vehicles. You need to create classes for Bus, Car and Truck. The methods fuelAmount(), capacity(), applyBrakes() will be same for all of the 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 in below figureClass or Super class

[



You can clearly see that 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. 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 rest of the three classes from base class(Vehicle).

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

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

[[Inheritance ka matlab hota hai ek class se dusri class bnana . Agar humare pass kuch aise class hai jiske ki kuch properties hume new class me chaiye to hum new class ko derive kar sakte hai using old class

Eg ke liye mann lo mere pass person class hai jisme ki name ,adhar no, address naam ke variables hai

Ab mujhe student class bnani hai jisme ki mjhe name ,adhar no, address, rollno,class etc naam ke variables chaiye toh hume person class se student class derive kar sakte hai

Yaha person class super class ya base class hogi

Aur student class sub class ya derived class hogi

Upar diye gaye eg me vehical naam ki base class se tin class derived ki gayi hai.

Iss tarah inheritance ka use karke hum code ko reuse kar sakte hai]]

Ek program ko samjhte hai

```
//sabse pehle ek parent naam ki class bnate hai jisme ek variable liya hai

class Parent
{
   public:
     int id_p;
};

//ab yha pe child naam ki class bnaye hai jo ki parent class se derive hogi

class Child : public Parent
{
   public:
     int id_c;
};

//main function
Void  main()
  {
```

```
	Child obj1;

	// An object of class child has all data members
	// and member functions of class parent
//child class ka object parent class ke pure members ko access kar sakegi
	obj1.id_c = 7;
	obj1.id_p = 91;
	cout << "Child id is " <<  obj1.id_c << endl;
	cout << "Parent id is " <<  obj1.id_p << endl;


  }
```

Iska output

```
Child id is 7
Parent id is 91
```

Modes of Inheritance

1. **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.

2. **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.

3. **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.

[inheritance tin takerko se ho sakta hai

Public ,protected and private]]

Niche eg me samjege

Jesa ki ap jante ho private sirf class ke andar public kahi bhi accessible ho sakta hai

Protected ka use inheritance m hota hai

Jis bhi variable ko hum protected declare krte hai wo sirf us class aur us se derived hone wali class m accessible ho sakta hai.

Class A ek class bnayi humne jisme three variables liye hai ek private type ka ek protected type ka aur ek public

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

Ab hum B class derive karege jo ki A se bani hai
Yaha public inheritance use ho rha hai

↘

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

Ab hum C class derive karege jo ki A se bani hai
Yaha protacted inheritance use ho rha hai

↘

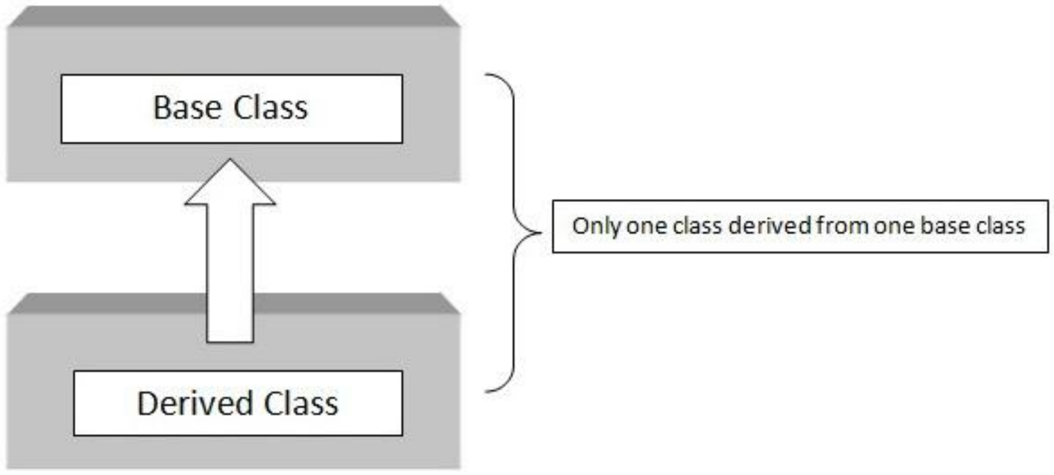
```
class C : protected A
{
   // x is protected
   // y is protected
   // z is not accessible from C
};
```
Ab hum D class derive karege jo ki A se bani hai
Yaha private inheritance use ho rha hai

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

Iss takerike se mode of inheritance decide karega ki variable kis type se class p access ho sakta hai

Types of Inheritance in C++

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.



Ism eek hi class ka use krke derive hoti hai class abhi tak humne jitney eg dekhe wo single inheritance ke the

Example

```
class Vehicle {
  public:
    Vehicle()
    {
      cout << "This is a Vehicle" << endl;
    }
};
```

```
class base    //single base class

{

   public:

     int x;
```

```
  void getdata()

  {

    cout << "Enter the value of x = "; cin >> x;

  }

 };

class derive : public base    //single derived class

{

  private:

   int y;

  public:

  void readdata()

  {

    cout << "Enter the value of y = "; cin >> y;

  }

  void product()

  {

    cout << "Product = " << x * y;

  }
```

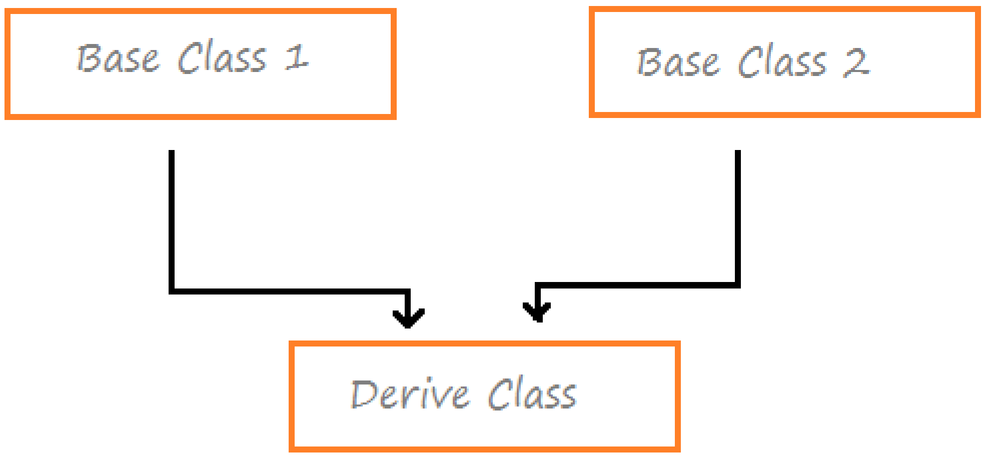
```
};



void main()

{

   derive a;     //object of derived class

   a.getdata();

   a.readdata();

   a.product();



}
```

Yha object derived class ka bnaya gya hai

Us object k thru hum base class ke aur derived class k members ko access kr sakte hai

2. **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**.



Multiple Inheritance

```
3. class subclass_name : access_mode base_class1, access_mode base_class2, ....
4. {
5.   //body of subclass
6. }
```

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

```
// multiple inheritance.cpp

class A

{

        public:

        int x;
```

```
            void getx()

    {

               cout << "enter value of x: "; cin >> x;

    }

};

class B

---------{

            public:

            int y;

            void gety()

            {

               cout << "enter value of y: "; cin >> y;

            }

};

class C : public A, public B   //C is derived from class A and class B

{

            public:

            void sum()
            {

               cout << "Sum = " << x + y;

            }

};



int main()

{

            C obj1; //object of derived class C

            obj1.getx();

            obj1.gety();

            obj1.sum();

            return 0;

}           //end of program
```

4. **Multilevel Inheritance**: If a class is derived from another derived class then it is called **multilevel inheritance**. So in C++ multilevel inheritance, a class has more than one parent class.

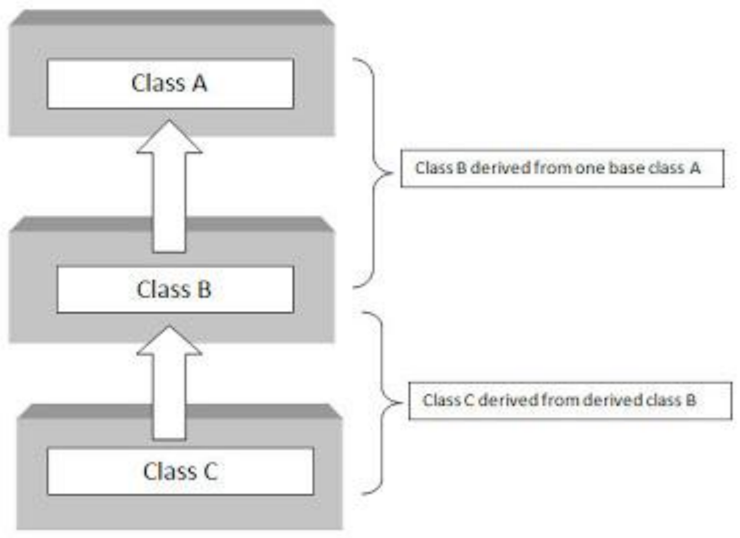
For example, if we take animals as a base class then **mammals** are the derived class which has features of animals and then humans are the also derived class that is derived from sub-class mammals which inherit all the features of mammals.

};

```
class base //single base class

{

            public:

            int x;

            void getdata()

            {

            cout << "Enter value of x= "; cin >> x;

            }

};

class derive1 : public base // derived class from base class

{

            public:

            int y;

            void readdata()

            {

               cout << "\nEnter value of y= "; cin >> y;

            }

};
```

```
class derive2 : public derive1   // derived from class derive1

{

              private:

              int z;

              public:

              void indata()

              {

              cout << "\nEnter value of z= "; cin >> z;

              }

              void product()

              {

                  cout << "\nProduct= " << x * y * z;

              }

};

int main()

{

     derive2 a;      //object of derived class

     a.getdata();
```

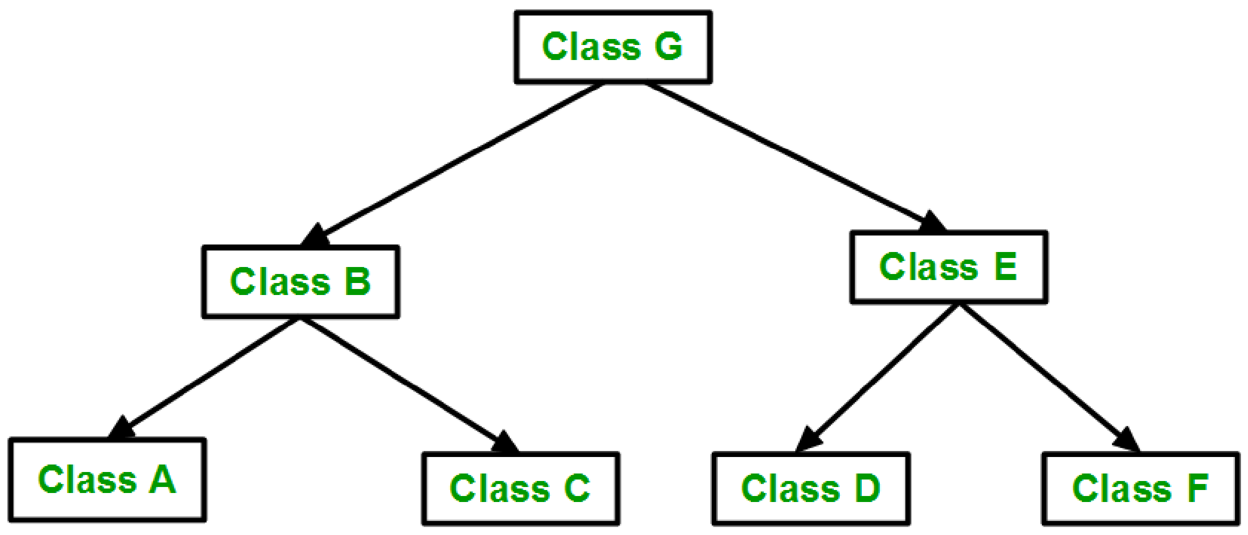
```
    a.readdata();

    a.indata();

    a.product();

    return 0;

}
```


Class A
Class B
Class C
Class B derived from one base class A
Class C derived from derived class B

4.Hierarchical Inheritance: When several classes are derived from common base class it is called **hierarchical inheritance**.

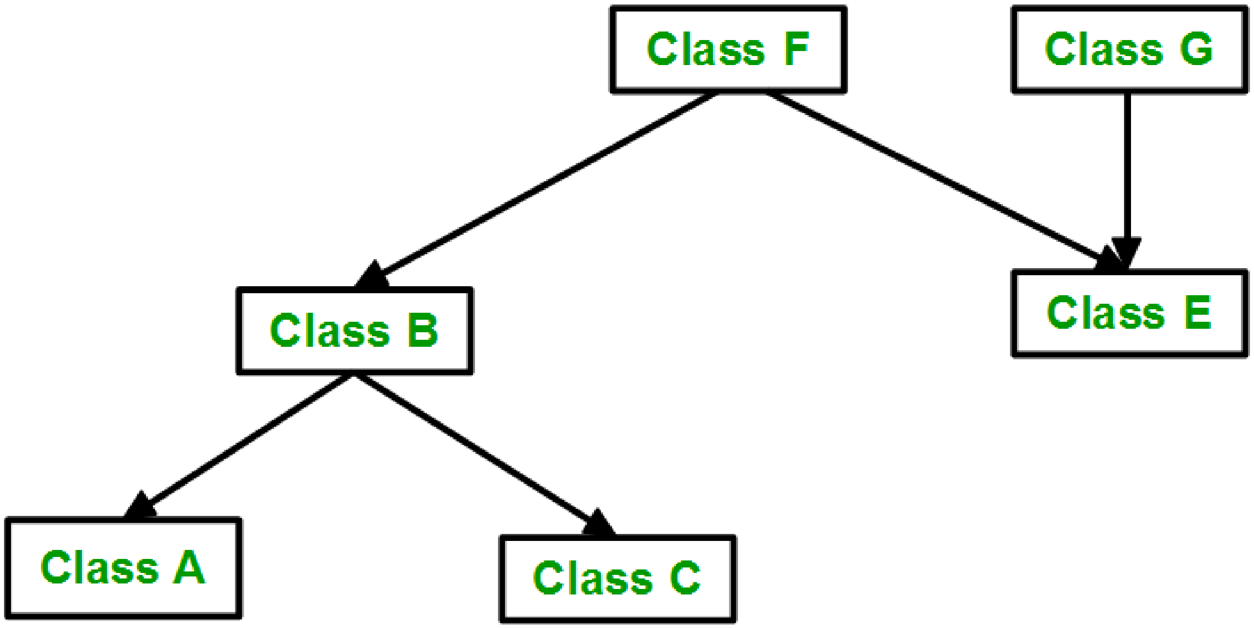
In **C++ hierarchical inheritance**, the feature of the base class is inherited onto more than one sub-class.

For example, a car is a common class from which Audi, Ferrari, Maruti etc can be derived.



```
class A //single base class

{

   public:

            int x, y;

            void getdata()

            {

               cout << "\nEnter value of x and y:\n"; cin >> x >> y;

            }

};

class B : public A //B is derived from class base

{
    public:

                void product()

                {

                    cout << "\nProduct= " << x * y;

                }

};

class C : public A //C is also derived from class base

{

    public:

                void sum()

                {

        cout << "\nSum= " << x + y;

                }

};

int main()

{

    B obj1;          //object of derived class B

    C obj2;          //object of derived class C
    obj1.getdata();

    obj1.product();

    obj2.getdata();

    obj2.sum();

    return 0;

}
```

5. **Hybrid 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 inheritance:



```
class A

{

            public:

            int x;

};

class B : public A

{

            public:

            B()     //constructor to initialize x in base class A

            {

              x = 10;

            }

};

class C

 {

            public:

            int y;

            C()   //constructor to initialize y
            {
                y = 4;
       }
};
class D : public B, public C   //D is derived from class B and class C
{
            public:
            void sum()
            {
                cout << "Sum= " << x + y;
            }
};

int main()
{
       D obj1;          //object of derived class D
            obj1.sum();
            return 0;
```

```
}
```

C++ virtual function

- A C++ virtual function is a member function in the base class that you redefine in a derived class. It is declared using the virtual keyword.
- It is used to tell the compiler to perform dynamic linkage or late binding on the function.
- There is a necessity to use the single pointer to refer to all the objects of the different classes. So, we create the pointer to the base class that refers to all the derived objects. But, when base class pointer contains the address of the derived class object, always executes the base class function. This issue can only be resolved by using the 'virtual' function.
- A 'virtual' is a keyword preceding the normal declaration of a function.
- When the function is made virtual, C++ determines which function is to be invoked at the runtime based on the type of the object pointed by the base class pointer.

Late binding or Dynamic linkage

In late binding function call is resolved during runtime. Therefore compiler determines the type of object at runtime, and then binds the function call.

```
class A
{
  int x=5;
   public:
   void display()
   {
       std::cout << "Value of x is : " << x<<std::endl;
   }
};
class B: public A
{
   int y = 10;
   public:
   void display()
   {
       std::cout << "Value of y is : " <<y<< std::endl;
   }
};
int main()
{
   A *a;
   B b;
   a = &b;
  a->display();
   return 0;
}
```

OUTPUT

```
Value of x is : 5
```

In the above example, * a is the base class pointer. The pointer can only access the base class members but not the members of the derived class. Although C++ permits the base pointer to point to any object derived from the base class, it cannot directly access the members of the derived class. Therefore, there is a need for virtual function which allows the base pointer to access the members of the derived class

C++ virtual function Example

Let's see the simple example of C++ virtual function used to invoked the derived class in a program.

```
#include <iostream>
{
 public:    1
 virtual void display()
 {
  cout << "Base class is invoked"<<endl;
 }
};
class B:public A
{
 public:
 void display()
 {
  cout << "Derived Class is invoked"<<endl;
 }
};
int main()
{
 A* a;    //pointer of base class
 B b;     //object of derived class
 a = &b;
 a->display();   //Late Binding occurs
}
```

Output:

```
Derived Class is invoked
```

Pure Virtual Function

- A virtual function is not used for performing any task. It only serves as a placeholder.
- When the function has no definition, such function is known as "**do-nothing**" function.
- The "**do-nothing**" function is known as a **pure virtual function**. A pure virtual function is a function declared in the base class that has no definition relative to the base class.
- A class containing the pure virtual function cannot be used to declare the objects of its own, such classes are known as abstract base classes.
- The main objective of the base class is to provide the traits to the derived classes and to create the base pointer used for achieving the runtime polymorphism.

Pure virtual function can be defined as:

```
virtual void display() = 0;
```

Let's see a simple example:

```
#include <iostream>
using namespace std;
class Base
{
    public:
    virtual void show() = 0;
};
class Derived : public Base
{
   public:
   void show()
   {
       std::cout << "Derived class is derived from the base class." << std::endl;
   }
};
int main()
{
   Base *bptr;
   //Base b;
   Derived d;
   bptr = &d;
   bptr->show();
   return 0;
}
```

Output:

```
Derived class is derived from the base class.
```

In the above example, the base class contains the pure virtual function. Therefore, the base class is an abstract base class. We cannot create the object of the base class.