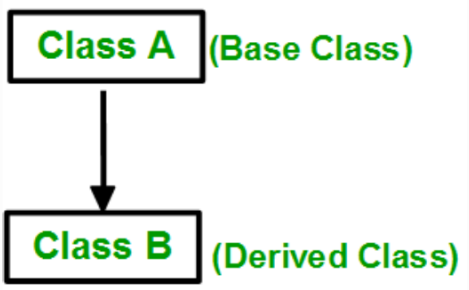
Types Of Inheritance:-

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

Types of Inheritance in C++

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.



Syntax:

class subclass_name : access_mode
base_class { // body of subclass }; OR
class A { ... . }; class B: public A { ... . };
// C++ program to explain

// Single inheritance

#include<iostream>

using namespace std;

```
// base class

class Vehicle {

  public:

    Vehicle()

    {

      cout << "This is a Vehicle\n";

    }

};



// sub class derived from a single base
classes
```

```
class Car : public Vehicle {


};


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

```
}

// 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:
```

```
#include<iostream>

using namespace std;



class A

{

    protected:

    int a;



    public:

        void set_A(int x)
```

```
    {
        a=x;
    }

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

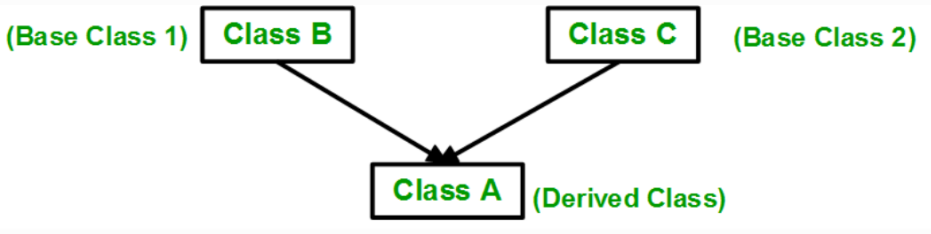
```
class B: public A

{

    int b,p;



    public:

        void set_B(int x,int y)

        {

            set_A(x);

            b=y;

        }
```

```
    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(4,5);
    _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**.



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

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

CPP

```
// 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;

}
```

OutputThis is a Vehicle This is a 4 wheeler Vehicle

```
#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
{
```

```
protected:
    int b;

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

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

class C: public B
{
    int c,p;
```

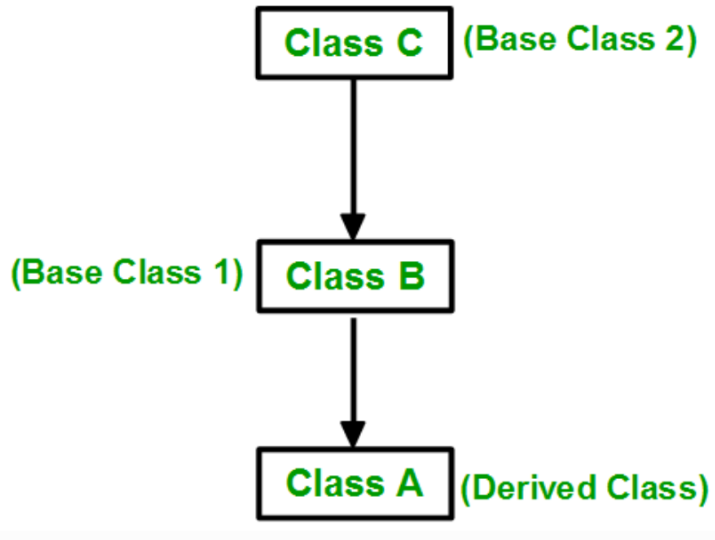
```
public:
    void set_C()
    {
        cout<<"Enter the Value of C=";
        cin>>c;
     }

     void disp_C()
     {
        cout<<endl<<"Value of C="<<c;
```

```
        }


        void cal_product()

        {

            p=a*b*c;

            cout<<endl<<"Product of "<<a<<"
* "<<b<<" * "<<c<<" = "<<p;

        }

};


main()
```

```
{


    C _c;

    _c.set_A();

    _c.set_B();

    _c.set_C();

    _c.disp_A();

    _c.disp_B();

    _c.disp_C();

    _c.cal_product();
```

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

CPP

```
// C++ program to implement

// Multilevel Inheritance

#include <iostream>

using namespace std;



// base class

class Vehicle {

public:
```

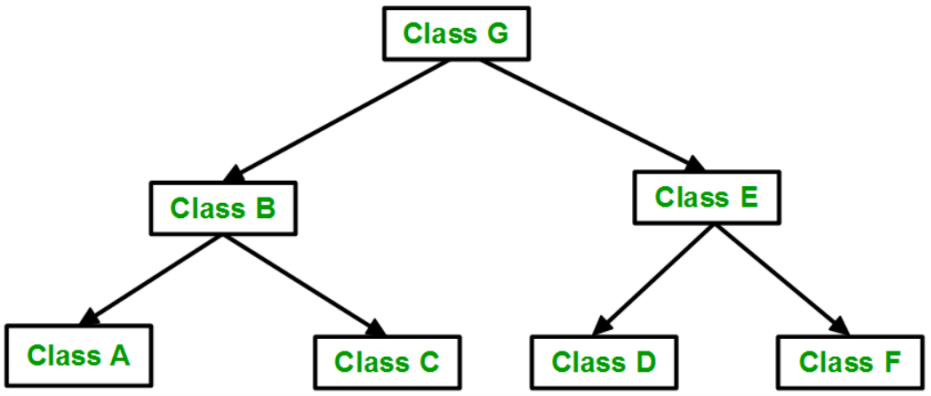
```
    Vehicle() { cout << "This is a Vehicle\n"; }

};



// 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;

}
```

OutputThis is a Vehicle Objects with 4 wheels are vehicles Car has 4 Wheels

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. }

CPP

// 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 {

};
```

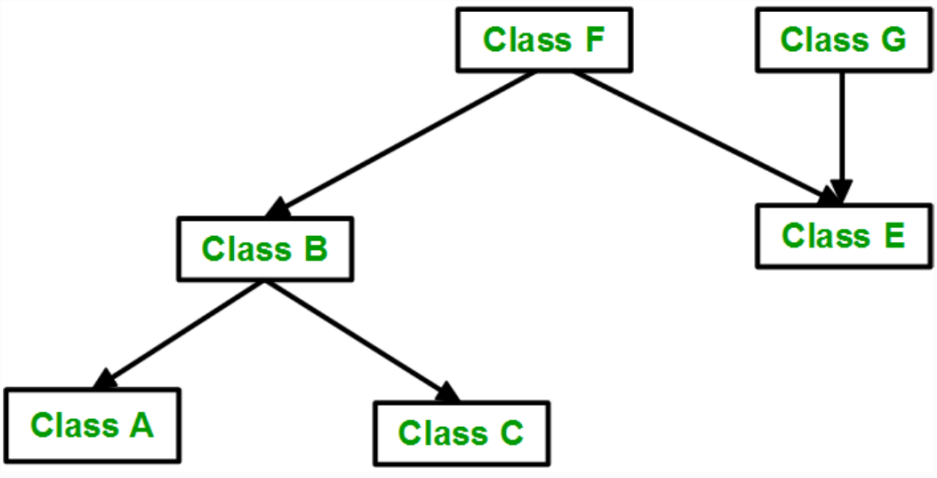
```
// second sub class
class Bus : public Vehicle {
};


// main function
int main()
{
    // Creating object of sub class will
    // invoke the constructor of base class.
```

```
    Car obj1;

    Bus obj2;

    return 0;

}
```

OutputThis is a Vehicle This is a Vehicle

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:



// 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;
}
```

OutputThis is a Vehicle Fare of Vehicle

C++

```
// 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;

}
```

6. A special case of hybrid inheritance: Multipath inheritance:

A derived class with two base classes and these two base classes have one common

base class is called multipath inheritance. Ambiguity can arise in this type of inheritance.

Example:

CPP

```
// C++ program demonstrating ambiguity
in Multipath

// Inheritance



#include <iostream>

using namespace std;
```

```
class ClassA {

public:

    int a;

};



class ClassB : public ClassA {

public:

    int b;

};



class ClassC : public ClassA {
```

```
public:

    int c;

};



class ClassD : public ClassB, public ClassC
{

public:

    int d;

};



int main()
```

```
{

    ClassD obj;



    // obj.a = 10;                  // Statement 1,
Error

    // obj.a = 100;                  // Statement 2,
Error



    obj.ClassB::a = 10; // Statement 3

    obj.ClassC::a = 100; // Statement 4
```

```
    obj.b = 20;

    obj.c = 30;

    obj.d = 40;



    cout << " a from ClassB  : " <<
obj.ClassB::a;

    cout << "\n a from ClassC  : " <<
obj.ClassC::a;



    cout << "\n b : " << obj.b;

    cout << "\n c : " << obj.c;

    cout << "\n d : " << obj.d << '\n';
```

}

Output a from ClassB : 10 a from ClassC : 100 b : 20 c : 30 d : 40

Output:

a from ClassB : 10 a from ClassC : 100 b : 20 c : 30 d : 40
In the above example, both ClassB and ClassC inherit ClassA, they both have a single copy of ClassA. However Class-D inherits both ClassB and ClassC, therefore Class-D has two copies of ClassA, one from ClassB and another from ClassC.
If we need to access the data member of ClassA through the object of Class-D, we must specify the path from which a will be accessed, whether it is from ClassB or ClassC, bcoz compiler can’t differentiate

between two copies of ClassA in Class-D.

There are 2 Ways to Avoid this Ambiguity:

1) Avoiding ambiguity using the scope resolution operator: Using the scope resolution operator we can manually specify the path from which data member a will be accessed, as shown in statements 3 and 4, in the above example.

CPP

```
obj.ClassB::a = 10;       // Statement 3

obj.ClassC::a = 100;      // Statement 4
```

Note: Still, there are two copies of ClassA in Class-D.

2) Avoiding ambiguity using the virtual base class:

CPP

```
#include<iostream>



class ClassA

{

  public:

    int a;

};
```

```
class ClassB : virtual public ClassA

{

  public:

    int b;

};



class ClassC : virtual public ClassA

{

  public:

    int c;
```

```
};


class ClassD : public ClassB, public ClassC
{
  public:
    int d;
};


int main()
{
    ClassD obj;
```

```
obj.a = 10;       // Statement 3

obj.a = 100;      // Statement 4


obj.b = 20;

obj.c = 30;

obj.d = 40;


cout << "\n a : " << obj.a;

cout << "\n b : " << obj.b;
```

```
    cout << "\n c : " << obj.c;

    cout << "\n d : " << obj.d << '\n';

}
```

Output:

a : 100 b : 20 c : 30 d : 40According to the above example, Class-D has only one copy of ClassA, therefore, statement 4 will overwrite the value of a, given in statement 3.