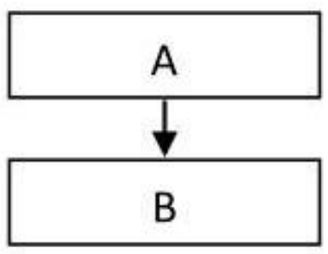
Unit-4 Inheritance

1 Explain Concept of Inheritance OR Explain type of inheritance with example.

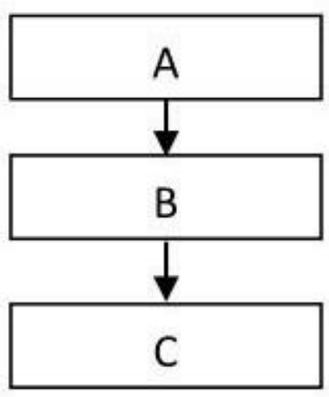
- Inheritance is the process, by which one class can acquire the properties and methods of another class.
- The mechanism of deriving a new class from an old class is called inheritance.
- The new class is called derived class and old class is called base class.
- The derived class may have all the features of the base class and the programmer can add new features to the derived class.
- **Type of inheritance :**

Single Inheritance



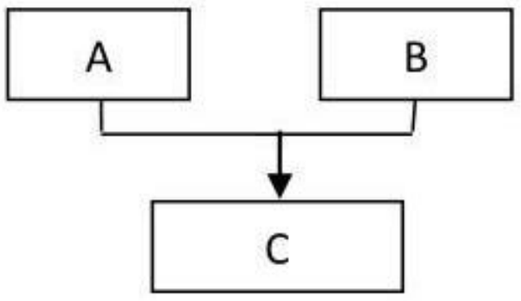
- If a class is derived from a single class then it is called single inheritance.
- Class *B* is derived from class *A*

Multilevel Inheritance



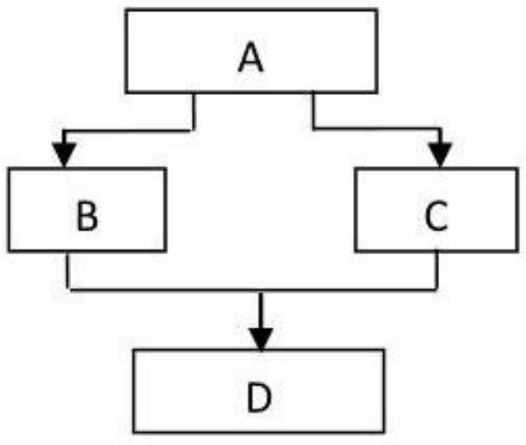
- A class is derived from a class which is derived from another class then it is called multilevel inheritance
- Here, class *C* is derived from class *B* and class *B* is derived from class *A*, so it is called multilevel inheritance.

Multiple Inheritance



- If a class is derived from more than one class then it is called *multiple inheritance*.
- Here, class *C* is derived from two classes, class *A* and class *B*.

Hybrid Inheritance



- It is a combination of any above inheritance types. That is either multiple or multilevel or hierarchical or any other combination.
- Here, class *B* and class *C* are derived from class *A* and class *D* is derived from class *B* and class *C*.
- Class *A*, class *B* and class *C* is example of Hierarchical Inheritance and class *B*, class *C* and class *D* is example of Multiple Inheritance so this hybrid inheritance is combination of Hierarchical and Multiple Inheritance.

2 Define Derived Class in inheritance.

- A derived class is defined by specifying it relationship with the base class in addition to its own details.
- General form of derived class is as shown below;

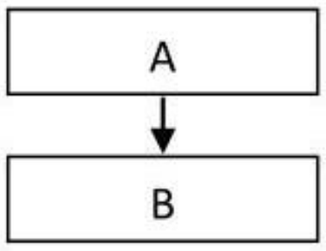
```
Class derived-class-name : visibility-mode base-class-mode
{
```

.....// members of derived class
.....//
}

- Where class is the required keyword, derived-class-name is the name given to the derived class, base-class-name is the name to the base class and ':' (colon) indicates that the derived-class-name is derived from the base-class-name, visibility-mode is optional and, if present, may be either private or public. The default visibility-mode is private. Visibility mode specifies whether the features of the base class are privately derived or publicly derived.

3 Explain Single Level Inheritance with example.

- Inheritance in which the derived class is derived from only one base class is called **single level inheritance**.



- **Syntax :**

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

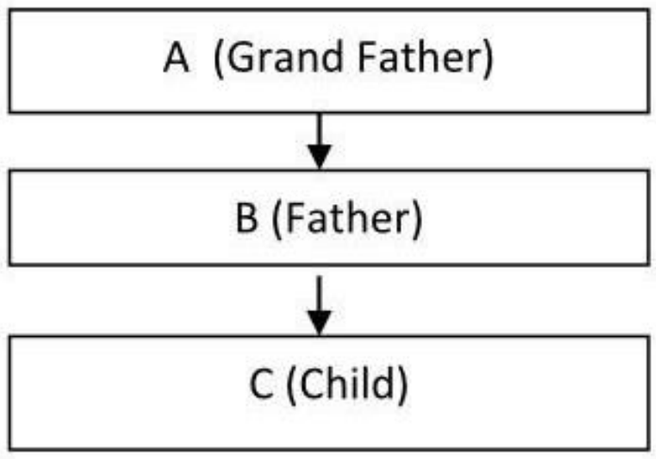
- **Example:**

```
#include<iostream>
using namespace std;
class A
{
        public:
                void methodA()
                {
                        cout<<"Hello, I am method of class A"<<endl;
                }
};
class B : public A
{
      public:
                void methodB()
                {
              cout<<"Hello, I am method of class B"<<endl;
                }
};
int main()
{
        B obj2;
        obj2.methodA();
        obj2.methodB();
        return 0;
}
```

Output:
"Hello, I am method of class A"
"Hello, I am method of class B"

4 Explain Multilevel Inheritance with example.

- In this type of inheritance, a derived class is created from another derived class.
- 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.




```
class A                     //Base class
{
        .....
};
class B : pullic A          //B derived from A
{
        .....
};
class C : public B          //C derived from B
{
        .....
};
```

Example :

```
#include<iostream>
using namespace std;
class A
{
        public:
                void m_1()
                {
                cout<<"Hello, I am method of Class A"<<endl;
                }
};
class B : public A
{
        public:
                void m_2()
                {
                cout<<"Hello, I am method of Class B"<<endl;
                }
};
```

```
class C : public B
{
        public:
                void m_3()
                {
                cout<<"Hello, I am method of Class C"<<endl;
                }
};
int main()
{
        C obj;
        obj.m_1();
        obj.m_2();
        obj.m_3();
        return 0;
}
```

Output :

"Hello, I am method of Class A"
"Hello, I am method of Class B"
"Hello, I am method of Class C"

Example 2 :

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

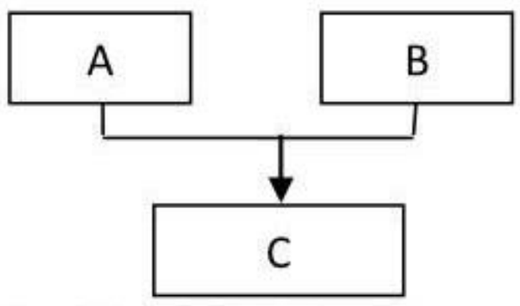
```
        {
        cout << "\nEnter value of z= "; cin >> z;
        }
        void product()
        {
           cout << "\nProduct= " << x * y * z;
        }
};
int main()
{
    derive2 a;     //object of derived class
    a.getdata();
    a.readdata();
    a.indata();
    a.product();
    return 0;
}
```

Output :

Enter value of x=2
Enter value of y=2
Enter value of z=2
Product=8

5 Explain Multiple Inheritance with example.

- If a class is derived from two or more base classes then it is called multiple inheritance. In **C++ multiple inheritance** a derived class has more than one base class.
- How does multiple inheritance differ from multilevel inheritance?
- **For example :**
- **Multilevel inheritance** : Inheritance of characters by a child from father and father inheriting characters from his father (grandfather)
- **Multiple inheritance** : Inheritance of characters by a child from mother and father



- **C++ Multiple Inheritance Syntax :**

```
class A
{
  .....
};
class B
{
  .....
 } ;
class C : acess_specifier A,access_specifier A // derived class from A and B
{
```

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

Example :

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

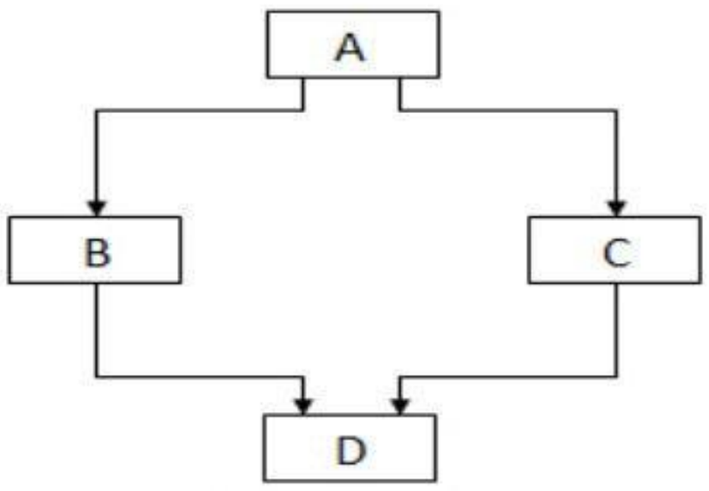
Output :

```
enter value of x:2
enter value of y:5
Sum =7
```

- In the above program, there are two base class A and B from which class C is inherited. Therefore, derived class C inherits all the public members of A and B and retains their visibility. Here, we have created the object obj1 of derived class C.

6 Explain Hybrid Inheritance with example.

- The inheritance in which the derivation of a class involves more than one form of any inheritance is called **hybrid inheritance**. Basically **C++ hybrid inheritance** is combination of two or more types of inheritance. It can also be called multi path inheritance.
- There could be situations where we need to apply two or more types of inheritance to design a program. At that time we have to apply more than one inheritance. So, this type of inheritance is known as the ***Hybrid Inheritance***



Hybrid Inheritance

- ***Syntax :***

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

Example :

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

Output :

Sum=14

Program 2 :

```
#include<iostream>
class A
{
        public:
                void m_1()
                {
                cout<<"Hello, I am mehtod of Class A"<<endl;
                }
};
class B : public A
{
        public:
                void m_2()
                {
                cout<<"Hello, I am mehtod of Class B"<<endl;
                }
};
class D
{
        public:
```

```
                void m_4()
                {
                cout<<"Hello, I am mehtod of Class D"<<endl;
                }
};
class C: public B,public D
{
        public:
                void m_3()
                {
                cout<<"Hello, I am mehtod of Class C"<<endl;
                }
};
int main()
{
        clrscr();
        C obj;
        obj.m_1();
        obj.m_2();
        obj.m_3();
        obj.m_4();
        return 0;
}
```

Output :

```
Hello, I am mehtod of Class A
Hello, I am mehtod of Class B
Hello, I am mehtod of Class C
Hello, I am mehtod of Class D
```

7 Making private data inheritable.

- We cannot inherit private data.
- We can inherit by making it public, but after making it public anyone can access from anywhere.
- C++ introduce new access modifier is **protected**.
- By making private data protected we can inherit it, but we cannot access outside.
- In between private and protected only one difference, private is not inheritable where as protected is inheritable

Specifiers	Within Same Class	In Derived Class	Outside the Class
Private	Yes	No	No
Protected	Yes	Yes	No
Public	Yes	Yes	Yes

Example :

```
#include <iostream>
using namespace std;
class A
{
        public:int a;
```

```
        protected:
        int b;
        public:
        int c;
        void init()
        {
                a=10;
                b=20;
                c=30;
        }
};
class B:public A
{
        public:
        void display()
        {
                cout<<a<<b<<c;
        }
};
int main()
{
        B bb;
        bb.init();
        bb.display();
}
```

Output :

102030

8 Explain virtual base class with example.

- It is used to prevent the duplication.
- In hybrid inheritance child class has two direct parents which themselves have a common base class.
- So, the child class inherits the grandparent via two separate paths. it is also called as indirect parent class.
- All the public and protected member of grandparent is inherited twice into child.
- We can stop this duplication by making **virtual base class**.

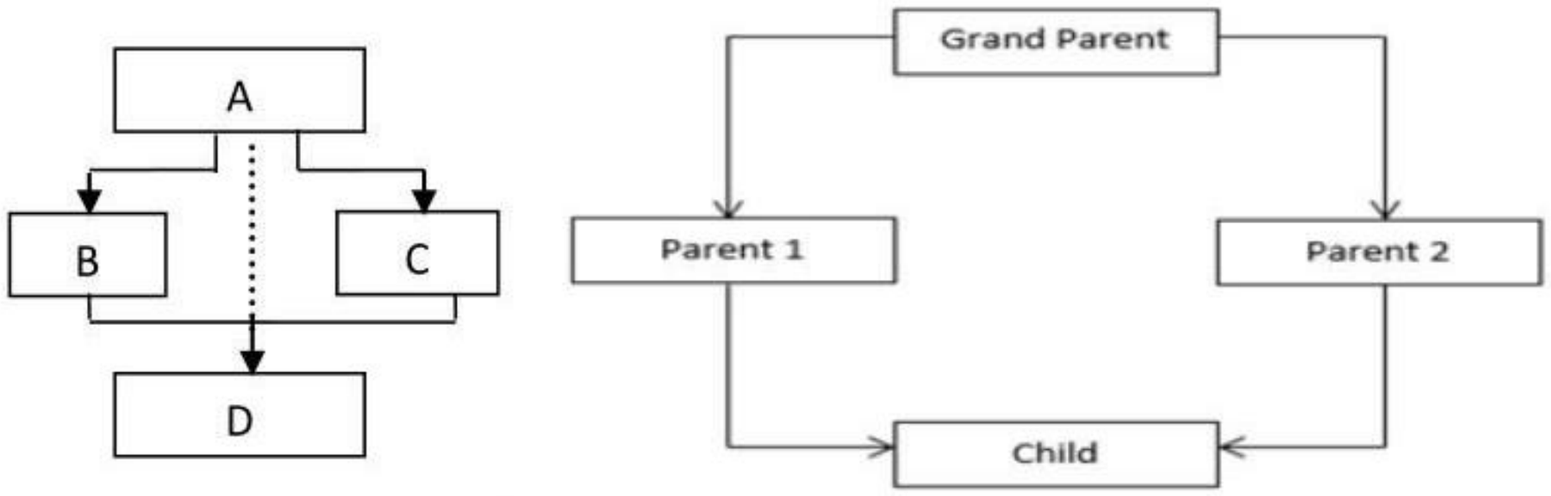


Figure: Multipath Inheritance

- As we can see from the figure that data members/function of class A are inherited twice to class D. One through class B and second through class C. When any data / function member of class A is accessed by an object of class D, ambiguity arises as to which data/function member would be called? One inherited

through B or the other inherited through C. This confuses compiler and it displays error.

Example Without using Virtual Base class :

```
#include <iostream>
using namespace std;
class A {
public:
    void show()
    {
            cout << "Hello form A \n";
    }
};

class B : public A {
};

class C : public A {
};

class D : public B, public C {
};

int main()
{
    D object;
    object.show();
}
```

Output :Error

```
prog.cpp: In function 'int main()':
prog.cpp:29:9: error: request for member 'show' is ambiguous
 object.show();
       ^
prog.cpp:8:8: note: candidates are: void A::show()
  void show()
       ^
prog.cpp:8:8: note: void A::show()
```

How to resolve this issue?

To resolve this ambiguity when class A is inherited in both class B and class C, it is declared as virtual base class by placing a keyword virtual as :

Syntax :

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

Example Using Virtual Base Class :

```
#include <iostream>
using namespace std;
class A {
public:
```

```
        void show()
        {
                cout << "Hello from A \n";
        }
};
class B : virtual public A {
};
class C : virtual public A {
};
class D : public B, public C {
};
int main()
{
        D object;
        object.show();
}
```

Output :

Hello from A

9 Explain abstract class.

- In C++ programming, sometimes inheritance is used only for the better visualization of data and you do not need to create any object of base class.
- For example: If you want to calculate area of different objects like: circle and square then, you can inherit these classes from a shape because it helps to visualize the problem but, you do not need to create any object of *shape*.
- In such case, you can declare *shape* as an abstract class. If you try to create object of an abstract class, compiler shows error.
- If expression **=0** is added to a virtual function then, that function is becomes pure virtual function.
- Note that, adding =0 to virtual function does not assign value, it simply indicates the virtual function is a pure function.
- If a base class contains at least one virtual function then, that class is known as abstract class.
- As shown in the following program, pure virtual function virtual float area () = 0; is defined inside class *Shape*, so this class is an abstract class and you cannot create object of class *Shape.*

Example :

```
#include<iostream>
using namespace std;
class shape
{
protected:
        float l;
public:
        void get_data()
        {
                cin>>l;
        }
```

```
                virtual float area()=0;
};
class square : public shape
{
        public:
                float area()
                {
                        return l*l;
                }
};
class circle : public shape
{
        public:
                float area()
                {
                        return 3.14*l*l;
                }
};

int main()
{
        square s;
        circle c;
        cout<<"Enter the lenght to calculate area of square:"<<endl;
        s.get_data();
        cout<<"Area of a square is: "<<s.area();
        cout<<"\nEnter radius to calculate area of a circle:";
        c.get_data();
        cout<<"Area of Circle: "<<c.area();
        return 0;
}
```

Output :

Enter the lenght to calculate area of square: 5
Area of a square is:25
Enter radius to calculate area of a circle:2
Area of Circle:12.56

10 Explain Constructors in derived class with example.

- Constructor is invoked automatically whenever an object of class is created, but in inheritance only derived class have object.
- So, whenever an object of derived class creates at that time first it will execute base class constructor then execute derived class constructor.
- If base class constructor have argument then we have to pass this argument from derived class constructor by following method

Syntax :

```
class A
{
```

```
    A(int a)
    {
        Statement 1;
        Statement 2;
         ...
        Statement n;
    }
};
class B:public A
{
     B(int x,int y):A(x)
    {
        Statement 1;
        Statement 2;
          ...
        Statement n;
      }
  };
```

- In above syntax we passed argument from derived class constructor to base class constructor.

Example :

```
#include <iostream>
using namespace std;
class A
{
        public:
        A(int a)
        {
                cout<<"\nValue of a="<<a;
        }
};
class B:public A
{
        public:
        B(int a,int b):A(a)
        {

                cout<<"\nValue of b="<<b;
        }
};
int main()
{
        B bb(10,20);
}
```

Output :

Value of a=10

Value of b=20