Unit-4
Inheritance

SYLLABUS

4.1 Concepts of inheritance

4.2 Defining derived classes

4.3 Single inheritance

4.4 Making a private member inherited

4.5 Multiple inheritance

4.6 Multilevel inheritance

4.7 Hybrid inheritance

4.8 Virtual base class

4.9 Abstract classes

4.10 Constructors in derived classes

4.1 Concepts of inheritance

- It is the process by which object of one class derived(acquried) the properties of object of another class.
- The mechanism of deriving a new class from an old class is called as inheritance.
- In inheritance, the old class is referred as the base class and the new class is called as the derived class or subclass.

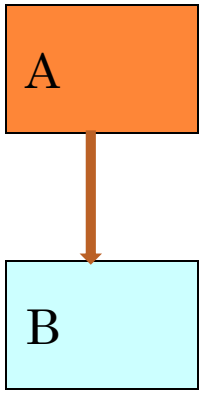
Inheritance

- A derived class with only one base class as single inheritance.
- A derived class with several base classes is called multiple inheritance.
- The traits of one class may be inherited by more than one class os called as hierarchical inheritance.
- The mechanism of deriving a class from another 'derived class' is called multiple inheritance.

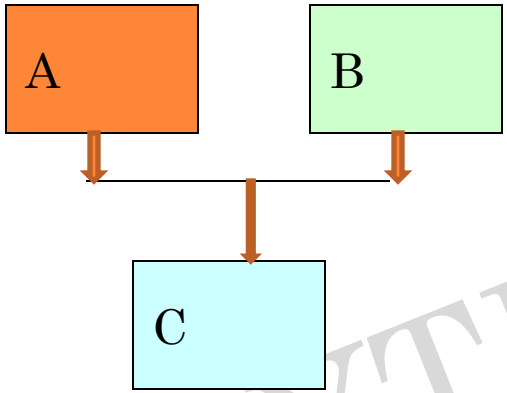
Types of inheritance

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

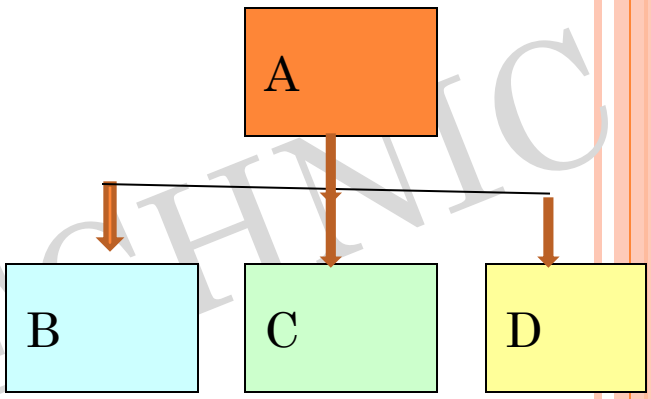
Forms of Inheritance



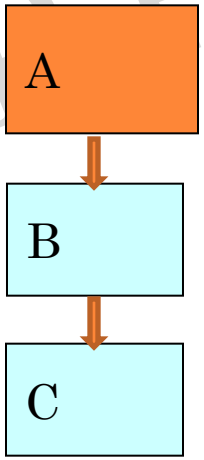
(a) Single Inheritance



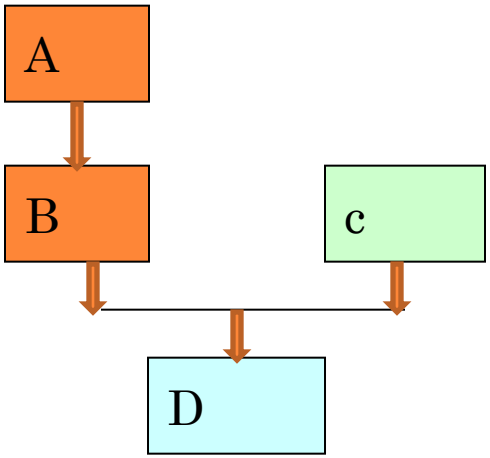
(b) Multiple Inheritance



(c) Hierarchical Inheritance



(a) Multi-Level Inheritance



(b) Hybrid Inheritance

4.2 Defining derived classes

A derived class can be defined by specifying its relationship with the base class.

Syntax:

Class derived_class_name: visibility_mode base_class_name

{

. . . . . .

. . . . . .

}

•The colon indicates that the derived-class-name is derived from the base-class-name.

•The **visibility mode** is optional. It may be either **public ,private, protected**.

•The default visibility mode is private.

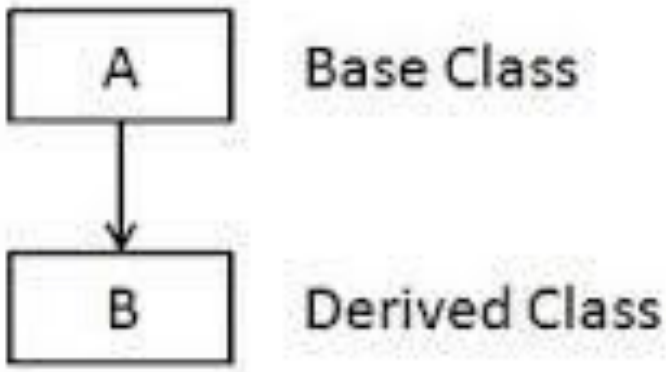
•Visibility mode specifies whether the features of the base class are privately derived, publicly derived or protected.

Examples:

```
class B : public A                    // public derivation
Members of B
}
class B : private A                   // private derivation
{
Members of B
}
class B : A                           // private derivation
{
Members of B
}
```

4.3 Single Inheritance

- In singlelevel inheritance there is only base class and one derived class.
- In figure, class A is called Base class and class B is called as derived class.



- Example of single inheritance:-

```
#include<conio.h>
#include<iostream.h>
class circle
{   protected:
            float a;
            int r;
    public:
    void getdata()
    {       cout<<"Enter r=";
            cin>>r;
    }
    void area()
    {       a=3.14*r*r;
            cout<<"Area="<<a;
    }       };
class circle_ext:public circle
{   protected:
            float p;
    public:
    void peri()
    {
            p=2*3.14*r;
            cout<<"Peripheral="<<p;
    }       };
void main()
{           clrscr();
            circle c1;          //base class
            c1.getdata();
            c1.area();
            circle_ext c2;      //derived
class
            c2.getdata();
            c2.area();
            c2.peri();
            getch();
}
```

Output:

```
Enter r=2
Area=12.56
Enter r=3
Area=28.26
Peripheral=18.84
```

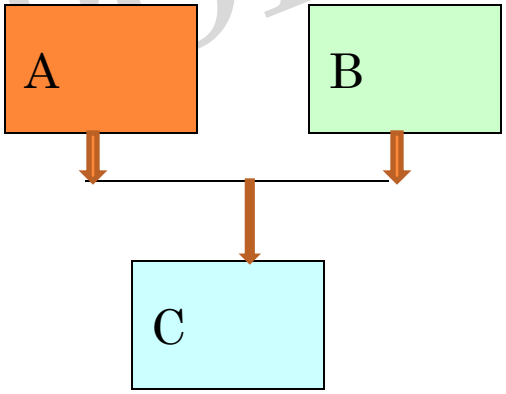
4.4 Making a private member inherited

- **Effect of visibility in Inheritance**

		Derived Class	
Base class	**Public Mode**	**Private Mode**	**Protected Mode**
Private	Not Inherited	Not Inherited	Not Inherited
Protected	Protected	Private	Protected
Public	Public	Private	Protected

4.5 Multiple Inheritance

- When two or more base classes are used from derivation of a class, it is called as multiple inheritance.
- In multiple inheritance, one derived class is derived from more than one base class.



Example of multiple inheritance

```
Class A
{
    -------
    -------
 };
Class B
{
   -------
   -------
};
Class c: public b , public a
{
   -------
   -------
};
```

```
#include<iostream.h>
#include<conio.h>
class num
{   protected:
    int a;
   public:
    num()
    {           a=0;            }
    num(int a1)
    {           a=a1;           }
    void display()
    {
        cout<<"a="<<a<<endl;
    }
};
class pair
{
 protected:
    int b,c;
 public:
        pair()
        {               b=0;        c=0;    }
        pair(int b1,int c1)
        {               b=b1;
                        c=c1;               }
        void display()
        {
            cout<<"b="<<b<<endl;
            cout<<"c="<<c<<endl;    }
};
 class triplet:public num,public pair
{
   public:
        triplet():num(),pair()
        {
        }
 triplet(int a1,int b1,int c1):num(a1),pair(b1,c1)
        {           }
        void display()
        {
cout<<"a="<<a<<"b="<<b<<"c="<<c<<endl;
        }
};
```

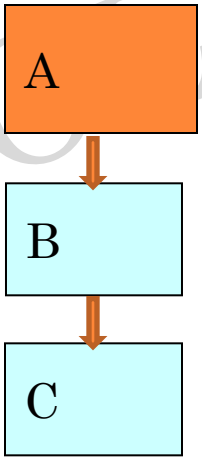
```
void main()
{
    clrscr();
    triplet t1(40,50,60);
    t1.display();
    getch();
}
```

Output:

a=40 b=50 c=60

4.6 Multilevel Inheritance

- When a class is derived from another derived clas i.e., derived class act as a base class, such type of inheritance is known as multilevel inheritance.
- In multilevel inheritance, the class is derived from another derived class.



- Example of Multilevel Inheritance :-

```
#include<conio.h>
#include<iostream.h>
class base
{   protected:
            int a;
    public:
            base()
            {           a=0;        }
            base(int a1)
            {           a=a1;       }
    void display( )
            {           cout<<"A="<<a;
            }
};
class derived1:public base
{   protected:
            int b;
    public:
            derived1()
            {           b=0;        }
derived1(int a1,int b1):base(a1)
                        {           b=b1;     }
                        void display()
                        {
            cout<<"A="<<a<<"B="<<b<<endl;
                        }
};
class derived2:public derived1
{           protected:
                        int c;
            public:
                        derived2()
                        {           c=0;      }
                        derived2(int a1,int b1,int
c1):derived1(a1,b1)
                        {           c=c1;     }
                        void display()
                        {
            cout<<"A="<<a<<"B="<<b
            <<"C="<<c;
                        }
};
```

```
void main()
{
   clrscr();
   base a1(10);
   a1.display();
   derived1 b1(20,30);
   b1.display();
   derived2 c1(40,50,60);
   c1.display();
   getch();
}
```

Output:

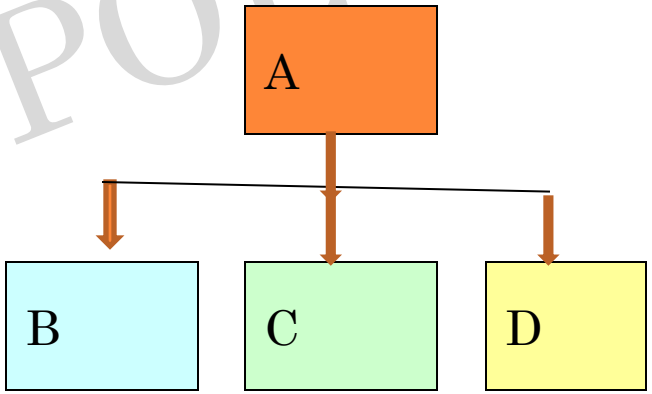
A=10

A=20 B=30

A=40 B=50 C=60

Hierarchical inheritance

- When a single base class is used for derivation of two or more classes it is known as hierarchical inheritance.
- In hierarchical inheritance, more than one derived class is derived from a single base class.



Example of hierarchical inheritance

```
Class A
{
  ------
  ------
};
Class B: public A
{
  ------
  ------
};
Class C:public A
{
 ------
 ------
};
```

```
#include<iostream.h>
#include<conio.h>
class vehicle
{   protected:
            double price;
    public:
            vehicle()
            {           price=0;            }
            vehicle(double p)
            {           price=p;            }
            void put_data()
            {
            cout<<"price="<<price<<endl;
            }
};
```

```
class water_vehicle:public vehicle
{           protected:
                        int speed;
public:

            water_vehicle():vehicle()
                        {
                                    speed=0;
                        }
                        water_vehicle(double
p,int s):vehicle(p)
                        {
                                    speed=s;
                        }
                        void put_data()
                        {

            cout<<"price(lacs)="<<price<<e
ndl;

            cout<<"speed="<<speed<<endl;
                        }
};
```

```
class road_vehicle:public vehicle
{   protected:
            int no;
    public:
            road_vehicle():vehicle()
            {           no=0;               }
            road_vehicle(double p, int
    n):vehicle(p)
            {           no=n;               }
            void put_data()
            {
            cout<<"price="<<price<<endl;
            cout<<"Chasis
    number="<<no<<endl;
            }
};
```

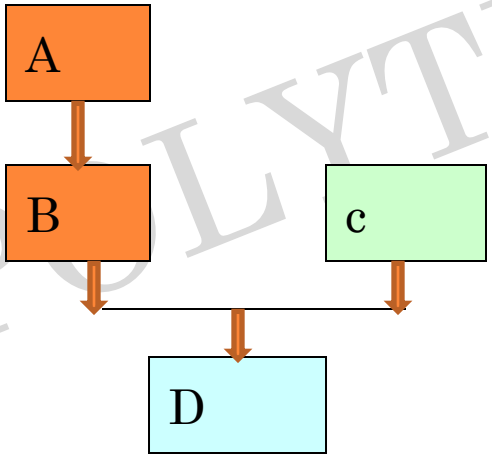
```
void main()
{
            water_vehicle
w1(20.45,25);
            cout<<"Water Vehicle-
>"<<endl;
            w1.put_data();
            cout<<endl;

            road_vegicle r1(12.34,45);
            cout<<"Road Vehicle-
>"<<endl;
            b1.put_data();
            cout<<endl;

            getch();
}
```

4.7 Hybrid inheritance

- The combination of more than one type of inheritance is known as hybrid inheritance.



Hybrid Inheritance

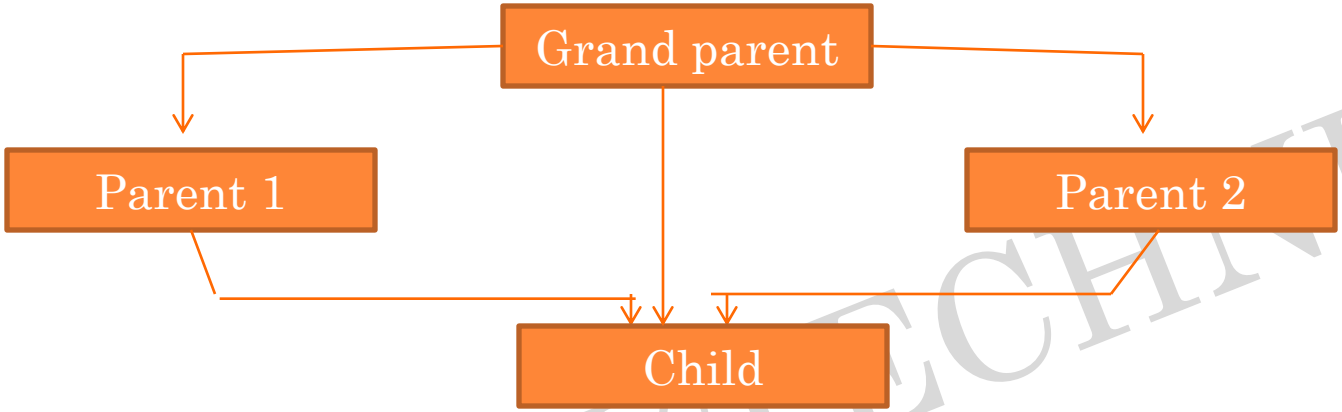
Example of hybrid inheritance

```
Class A
{
 ------
 ------
};
Class B:public A
{
 ------
 ------
};
Class C:public A
{
  ------
  -----
};
Class D:public B, public C
{
  ------
  ------
};
```

```
#include<iostream.h>
#include<conio.h>
class vehicle
{   protected:
    double  price;
    public:
            vehicle()
            {           price=0;            }
            vehicle(double p)
            {           price=p;            }
            void put_data()
            {
            cout<<"price="<<price<<endl;
            }
};
class water_vehicle:virtual public
    vehicle
{   protected:
            int speed;
public:
            water_vehicle():vehicle()
            {           speed=0;            }
            water_vehicle(double p,int
s):vehicle(p)
            {           speed=s;            }
            void put_data()
            {
            cout<<"price="<<price<<endl;
            cout<<"speed="<<speed<<endl;
            }
};
class road_vehicle:virtual public vehicle
{           protected:
                        int no;
            public:
            road_vehicle():vehicle()
            {           no=0;               }
road_vehicle(double p, int n):vehicle(p)
            {           no=n;               }
void put_data()
{     cout<<"price(lacs)="<<price<<endl;
      cout<<"Chasis number="<<no<<endl;
} };
```

```
class w_r_vehicle:public water_vehicle,public road_vehicle
{
    public:
            w_r_vehicle(double p,int s,int n):
    water_vehicle(p,s),road_vehicle(p,n)
            {
                    no=n;
            }
            void put_data()
            {       cout<<"price(lacs)="<<price<<endl;
                    cout<<"speed="<<speed<<endl;
                    cout<<"Number="<<no<<endl;
            }
};
 void main()
{   w_r_vehicle w1(20.45,25,40);
    cout<<"Water Road Vehicle->"<<endl;
    w1.put_data();
    getch();
}
```

4.8 Virtual Base Class



- The child class has two direct base class parent1 and parent2, which themselves have a common base class grandparent.
- The child class inherits all data of grandparent via two separate paths.
- It can also directly inherit as shown by line.

Example of virtual base class

```
class grandparent
{
};
class parent1:virtual public grandparent
{
};
class parent2:virtual pubic grandparent
{
};
class child: public parent1,public parent 2
{
};
```

```
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;
};

void  main()
{
   D ob;
   ob.i = 10;    //only one copy of i is
inherited.
   ob.j = 20;
   ob.k = 30;
   ob.sum = ob.i + ob.j + ob.k;
   cout << "Sum is : "<< ob.sum <<"\n";

   }
```

Output:

Sum is :60

4.9 ABSTRACT CLASSES

- A class from which we never want to create objects is called an abstract class.
- Such class exist only as a parent for derived classes.
- If we try to create object of such base class then compiler would generate erroe.

4.10 CONSTRUCTOR IN DERIVED CLASSES

- Base class constructors are always called in the derived class constructors.
- Whenever you create derived class object, first the base class default constructor is executed and then the derived class's constructor finishes execution.
- To call base class's parameterized constructor inside derived class's parameterized constructor, we must mention it explicitly while declaring derived class's parameterized constructor.

Example:

```
#include<conio.h>
#include<iostream.h>
class base
{   protected:
            int a;
    public:
            base()
            {           a=0;                    }
            base(int a1)
            {           a=a1;                   }
            void display()
            {
                        cout<<"A="<<a;
            }
};
class derived:public base
{   protected:
            int b;
public:
            derived() // Constructor in
Derived Class
                        {           b=0;      }
            derived(int a1,int b1):base(a1)  ////
Constructor in Derived Class
            {           b=b1;                 }
            void display()
            {

            cout<<"A="<<a<<"B="<<b<<endl;
            }
};
void main( )
{
            clrscr();
            base a1(10);
            a1.display();
            derived  b1(20,30);
            b1.display();
}
```