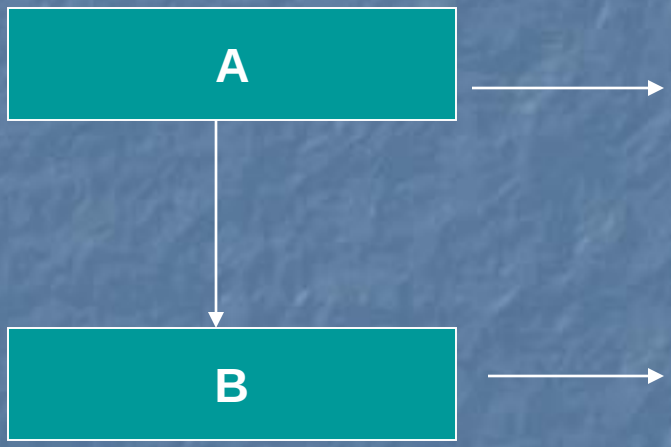
Inheritance

It is the mechanism of deriving new class from existing class.It provides the idea of reusability.



Base , super,parent class

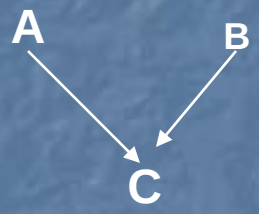
derived , sub, child class

Types of inheritance

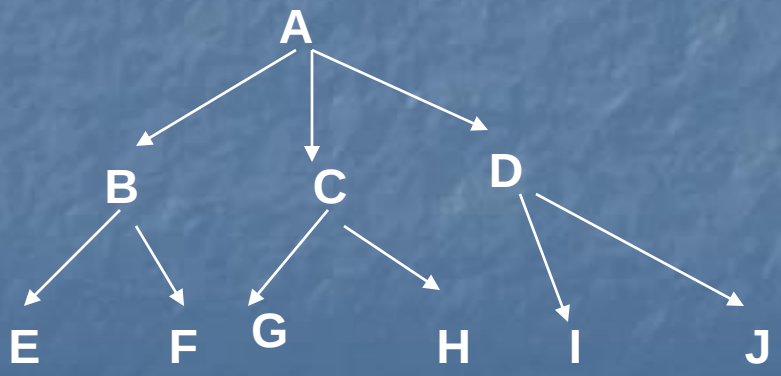
Single level

Multilevel

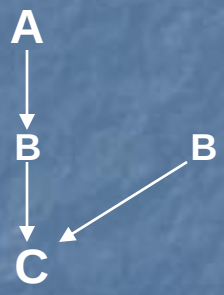
Multiple

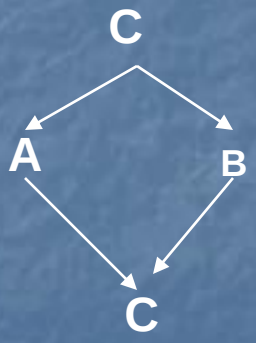


Hierarchical



hybrid





Public members provides lowest security and it can be accessed in whole programme.

Private members are visible only to class.It provides the highest security.

Protected members has middle security and it is visible to the class and classes derived from class. It is not accessible in main

syntax

- Class classname: mode oldclassname

Mode:

- Public : inherit all member as in base class mode(public to public and private to private).
- Private: all members of base class become private to derived class.
- Protected: all members of base class become protected to derived class.

```
Class base
{
Private:
Int x;
Protected:
Int y;
Public:
Int z;
};
Class der1:public base
{
Public:
Void get()
{
Int o;
o=x;    //error not accessible
o=y;    // ok
o=z;    //ok
}
};
```

```
Class der2:private base
{
Public:
Void get()
{
Int p;
p=x;      //error not accessible
p=y;      // ok
p=z;      //ok
}
};

Void main()
{
Int M;
der1 d1;
M=d1.x;              //error
M=d1.y;              //error
M=d1.z;              //ok
der2 d2;
M=d2.x;      //error
M=d2.y;      // error
M=d2.z;      //error
}
```

Single level inheritance

```
#include<iostream.h>
Class base
{
Public:
display()
{
Cout<<"It is base class";
}
};
Class derived:public base
{
};
Void main()
{
Derived d;
d.display();
}
```

```
#include<iostream.h>
Class base
{
Int x;
Public:
Get_base(int a)
{
X=a;
}
Display_base()
{
Cout<<"x="<<x;
}
};
```

```
Class derived:public base
{
Int y;
Public:
Get_derived(int a)
{
y=a;
}
Display_derived()
{
Cout<<"y="<<y;
}
};

Void main()
{
derived d;
d.Get_base(10);
d.Dispaly_base();
d.Get_derived(20);
d.Dispaly_derived();
}
```

```
#include<iostream.h>
Class base
{
int x;
Public:
Get_base(int a)
{
X=a;
}
Display_base()
{
Cout<<"x="<<x;
}
};
```

```
Class derived:public base
{
Int y;
Public:
Get_derived(int a)
{
Get_base(40);
y=a;
}
Display_derived()
{
Display_base();
Cout<<"y="<<y;
}
};

Void main()
{
derived d;
d.Get_derived(10);
d.Dispaly_derived();
}
```

Private inheritance

```
#include<iostream.h>
Class base
{
Public:
Display_base()
{
Cout<<"It is base
   class";
}
};
```

```
Class derived:private base
{
};
Void main()
{
derived d;
d.Dispaly_base();
}
```

o/p ------ error

```
Class base
{
Int x;
Public:
Get_base(int a)
{
X=a;
}
Display_base()
{
Cout<<"x="<<x;
}
};
```

```
Class derived:private base
{
Int y;
Public:
Get_derived(int a)
{
Get_base(20);
y=a;
}
Display_derived()
{
Display_base();
Cout<<"y="<<y;
}
};

Void main()
{
derived d;
d.Get_derived(10);
d.Dispaly_derived();
}
```

Function overriding

```
#include<iostream.h>
Class base
{
Int x;
Public:
Getdata(int a)
{
X=a;
}
Display ()
{
Cout<<"x="<<x;
}
};
```

```
Class derived:public base
{
Int y;
Public:
Getdata(int a)
{
y=a;
}
Display()
{
Cout<<"y="<<y;
}
};

Void main()
{
derived d;
d.Getdata();
d.Dispaly();
}
```

o/p--- y =

```
#include<iostream.h>
Class area
{
Protected:
Int length,width;
Public:
Void getdata()
{
Cout<<"enter length and width".;
Cin>>length>>width;
}
};
Class volume :public area
{
Int height;
Public:
Void get()
{
Getdata();
Cout<<"enter height";
Cin>>height;
}
Void cal_volume()
{
Cout<<"volume="<<length*height*width;
}
};
void main()
{
Volume v;
v.get();
v.cal_volume();
}
```

```
#include<iostream.h>
Class area
{
Protected:
Int length,width;
Public:
Void getdata()
{
Cout<<"enter len and width";
}
};
Class volume :public area
{

Public:
Int height;
Void get()
{
Getdata();
Cout<<"enter height";
Cin>>height;
}
};
```

```
void main()
{
Volume v;
v.get();
Int vol=length*height*width;
Cout<<"volume="<<vol;
}
```

o/p === error (protected member can be accessed in that class and in derived class not in main)

Multilevel inheritance

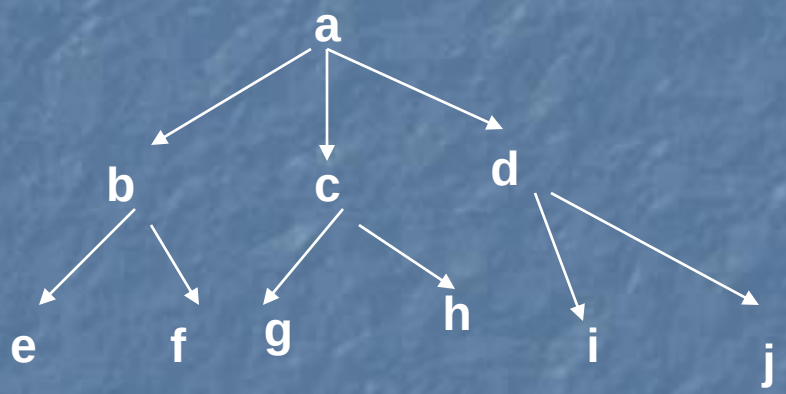
```
#include<iostream.h>
Class first
{
Public:
Display1()
{
Cout<<“first class”;
}
};
Class second:public first
{
Public:
Display2()
{
Cout<<“second class”;
}
};
Class third:public second
{
Public:
Display3()
{
Cout<<“third class”;
}
};
Void main()
{
third t;
t.Display1();
t.Display2();
t.Display3();

}
```

Multiple inheritance

```
#include<iostream.h>
Class super
{
Protected:
int base;
Void getbase()
{
Cout<<"enter base";
Cin>>base;
}
};
Class sub:public super
{
Protected:
Int power:
Void getpower()
{
Cout<<"enter power";
Cin>>power;
}
};
```

```
Class result:public super,public:sub
{
Public:
Void show()
{
getbase();
getpower();
Int t=1;
for(int i=1;i<=power;i++)
{
t=t*base;
}
Cout<<"base="<<base;
Cout<<"power="<<power;
Cout<<"output result="<<t;
}
};

Void main()
{
Result t;
t.show();
```

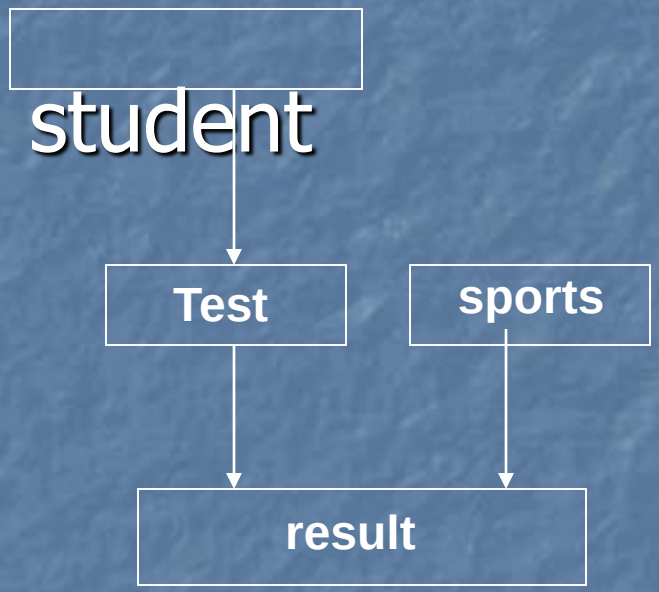
Hierarchical inheritance

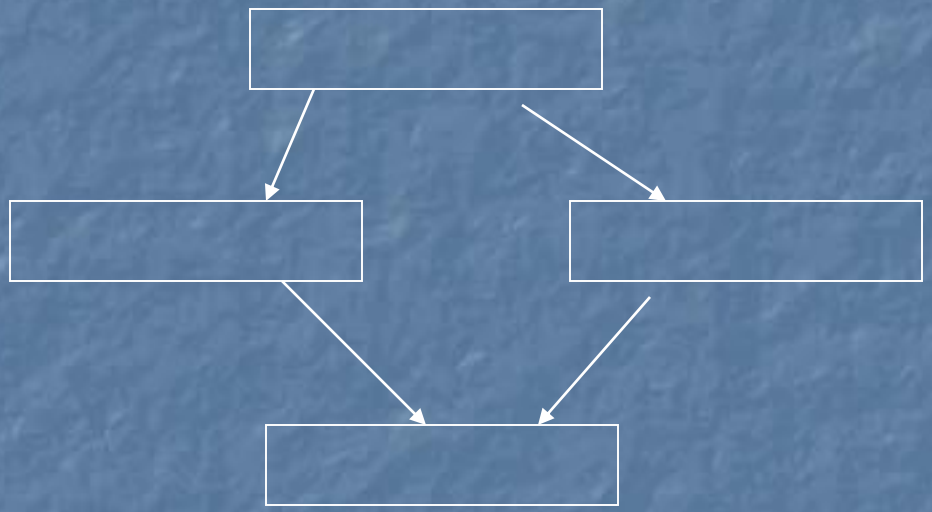


```
#include<iostream.h>
#include<string.h>
Class university
 {
Protected:
Char uname[20];
Public:
   void getuniversity()
    {
   Strcpy(uname,"RTU");
    }
 };

Class college1: public university
{
Protected:
Char college_name[20]
Public:
   college1()
   {
   Strcpy(college_name,"I.E.T");
   }
Void display1()
{
Cout<<"college
     name=<<college_name;
Cout<<"university name=<<uname;
}
```

```
Class college2: public university
{
Protected:
Char college_name[20];
Public:
   college2()
{
   Strcpy(college_name,"A.I.E.T");
   }
Void display2()
{
Cout<<"college name=<<college_name;
Cout<<"university name=<<uname;
}
  };
Void main()
{
University u;
u.getuniversity();
College1 c1;
C1.display1();
College2 c2;
C2.display2();
}
```

Hybrid





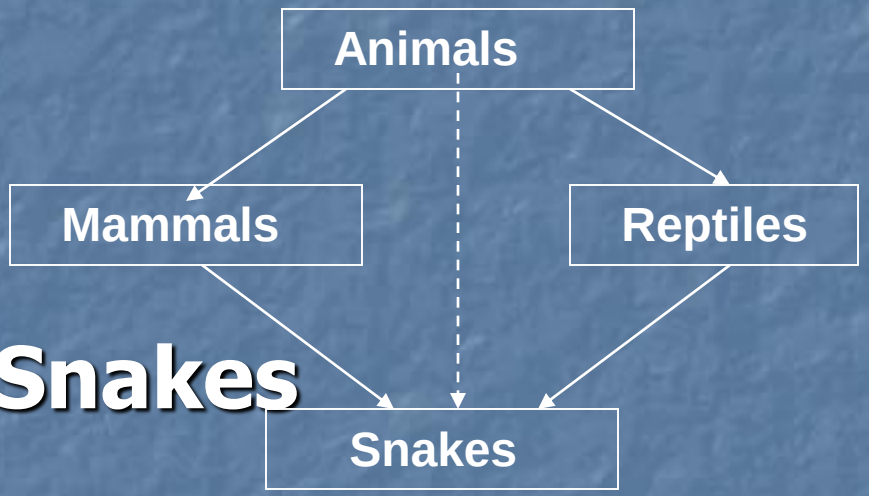
```
class student
{
private :
int rn;
char na[20];
public:
void getdata()
{
cout< < "Enter Name And Roll No : ";
cin>>na>>rn;
}
void putdata()
{
cout< < endl< < na< < "\t"< < rn< <
"\t";
}
};

class test : public student
{
protected:
float m1,m2;
public:
void gettest()
{
cout< < endl< < "Enter your marks In
CP 1 And Cp 2 :";
cin>>m1>>m2;
}
void puttest()
{
cout< < m1< < "\t"< < m2< < "\t";
}
};
```

```
class sports
{
protected:
float score;
public:
void getscore()
{
cout< < endl< < "Enter your score :";
cin>>score;
}
void putscore()
{
cout< < score< < "\t";
}
};
class results : public test , public sports
{
private :
float total;
public :
void putresult()
{
total = m1+m2+score;
cout< < total;
}
};
```

```
void main()
{
results r[5];
clrscr();
for(int i=0;i< 5;i++)
{
r[i].getdata();
r[i].gettest();
r[i].getscore();
}
cout< < endl< < "Name\tRollno\tCP 1\tCP
2\tScore\tTotal“; for(i=0;i< 5;i++)
{
r[i].putdata();
r[i].puttest();
r[i].putscore();
r[i].putresult();
}
getch();
}
```

2nd example of hybrid:

- PROBLEM:?
- Animals->Mammals
- Animals>Reptiles
- Mammals, Reptiles **->Snakes**



- **SO DUPLICATION FROM B AND C TO D CLASS SO IT CONFUSES THE COMPILER AND IT DISPLAYS ERROR**

SOLUTION OF IT: VIRTUAL

```
class a                    //ERROR
{
};
class b:public a
{ };
class c:public a
{ };
class d: public b, public c
{ };
```

VIRTUAL BASE CLASS

```
class a
{
};
class b:public virtual a
{
};
class c:virtual public a
{
};
class d: public b, public c
{ };
```

B and C share the same subobject of A. Using the keyword virtual in this example ensures that an object of class D inherits only one subobject of class A

```
Class a
{
Public:
Void show( )
{
Cout<<"It is base class";
}
};
Class b:virtual public a
{ };
Class c:virtual public a
{ };
Class d:public  b,public c
{ };
```

```
Void main()
{
D obj;
Obj.show();
}
```

o/p: It is base class

Constructor and destructor in inheritance

- Constructor: first base class then derived class constructor is called.
- Destructor: Reverse(first derived class then base class destructor is called).

Constructor and destructor in inheritance

```
Class base
{
Public:
Base()
{
Cout<<"it is base class<<endl";
}
};
Class derived:public base
{
Public:
derived()
{
Cout<<"it is derived class";
}
```

```
Void main()
{
Derived d;
}
```

O/P

it is base class

it is derived class

Destructor in inheritance

- Called in reverse order

```
Class base
{
Public:
Base()
{
Cout<<"it is base class constructor<<endl";
}
~Base()
{
Cout<<"it is base class destructor<<endl";
}

};
Class derived:public base
{
Public:
derived()
{
Cout<<"it is derived class
    constructor<<endl";
}
~derived()
{
Cout<<"it is derived class
    destructor<<endl";
}
```

```
Void main()
{
Derived d;
}
```

o/p :

it is base class constructor
it is derived class constructor
it is derived class destructor
it is base class destructor

Explicitly calling constructor

```
Class base
{
Public:
Base()
{
Cout<<"it is base class
    constructor<<endl";
}
~Base()
{
Cout<<"it is base class destructor<<endl";
}

};
Class derived:public base
{
Public:
derived() :base()
{
Cout<<"it is derived class
    constructor<<endl";
}
~derived()
{
Cout<<"it is derived class
destructor<<endl";
}
};
Void main()
{
Derived d;
}
```

o/p :

it is base class constructor
it is derived class constructor
it is derived class destructor
it is base class destructor

Constructor and destructor in multilevel

```
Class base
{
Public:
Base()
{
Cout<<"it is base class
    constructor<<endl";
}
~Base()
{
Cout<<"it is base class destructor<<endl";
}

};
Class derived1:public base
{
Public:
derived1()
{
Cout<<"it is derived1 class
    constructor<<endl";
}
~derived1()
{
Cout<<"it is derived1 class
    destructor<<endl";
}
};
```

```
Class derived2:public derived1
{
Public:
derived()
{
Cout<<"it is derived2 class constructor
<<endl";
}
~derived()
{
Cout<<"it is derived2 class destructor
<<endl";
}
};

Void main()
{
Derived2  obj;
}
```

o/p:

```
it is base class constructor
it is derived1 class constructor
it is derived2 class constructor
it is derived2 class destructor
it is derived1 class destructor
it is base class destructor
```

Constructor and destructor in multiple inheritance

Constructor is called in the sequence of inheritance

```
Class  a
{ };
Class b
{ };

Class c: public b,public a
{
};
```

Then first constructor of b then of a is called

```
Class base1
{
Public:
Base()
{
Cout<<"it is base1 class
    constructor<<endl";
}

};
Class base2
{
Public:
base2()
{
Cout<<"it is base2
 class constructor<<endl";
}
};
```

```
Class derived:public base2,public base1
{
Public:
derived()
{
Cout<<"it is derived class constructor
<<endl";
}
};

Void main()
{
Derived d;
}
```

o/p:

it is base2 class constructor
it is base1 class constructor
it is derived class constructor

```
Class base1
{
Public:
Base()
{
Cout<<"it is base1 class
   constructor<<endl";
}

};
Class base2
{
Public:
base2()
{
Cout<<"it is base2
 class constructor<<endl";
}
};
```

```
Class derived:public base2,public base1
{
Public:
derived():base1(),base2()
{
Cout<<"it is derived class constructor
<<endl";
}
};

Void main()
{
Derived d;
}
```

o/p:
it is base2 class constructor
it is base1 class constructor
it is derived class constructor

Same output even done explicit calling

```
Class base
{
Int x;
Public:
base(int a)
{
X=a;
Cout<<"x="<<x<<endl;
}};
Class derived:public base
{
 int y;
Public:
derived(int l,int m):base(l)
{
y=m;
Cout<<"y="<<y;
}
};
```

```
Void main( )
{
Derived d(20,30);
}
```

o/p

X=20

Y=30

Initialize the values of data members from constructor

```
Class demo
{
 int x,y;
Public:
demo(int l,int m):x(l),y(m)
{
Cout<<"x="<<x<<endl<<"y="<<y;
}
};
```

```
Void main()
{
Demo d;
}
```

Containership

- If we create the object of one class into another class and that object will be a member of the class.then it is called containership.

- This relation is calles has_a relation.
- While the inheritance is called kind_of and is_a
- Relation.

```
Class upper
{
Public:
Void display()
{
Cout<<"hello"<<endl;
}
};
Class  lower
{
Upper obj;
Public:
Lower()
{
Obj.display();
}

};
```

```
Void main()
{
Lower l;
}
```

o/p

hello

Container class also call upper class constructor first like inheritance.It can use only public member of upper class not the private and protected members

```
Class upper
{
Public:
upper()
{
Cout<<"it is upper class constructor
<<endl";
}

};
Class lower
{
Upper obj;
Public:
lower()
{
Cout<<"it is lower class constructor
<<endl";
}
};
```

```
Void main()
{
lower l;
}
```

o/p

it is upper class constructor
it is lower class constructor