CHAPTER 3

INHERITANCE: Concept of Reusability

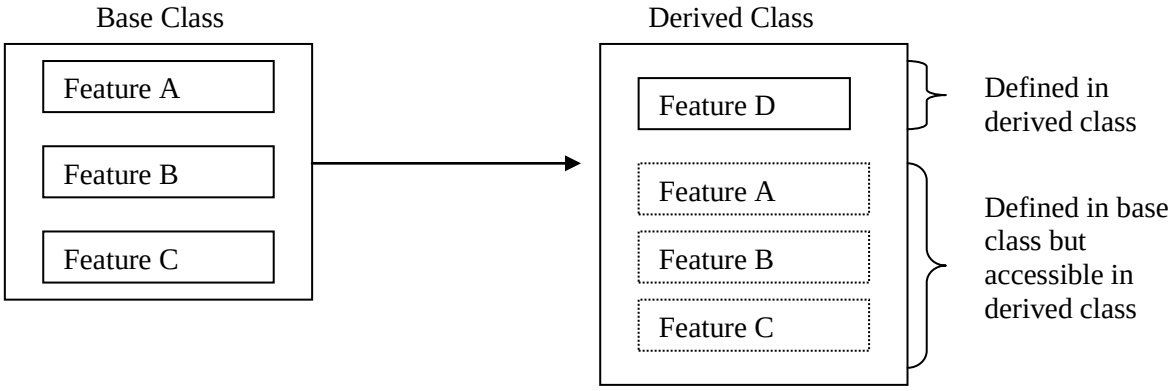
Objectives: **Marks** 20

- Concept of Inheritance & its types.
- Types of Visibility modes.
- Programs based on Inheritance.

4.1 Introduction, defining a derived class, visibility modes &effects.
4.2 Types of Inheritance: Single, multilevel, multiple, hierarchical, hybrid
4.3 Virtual base class, abstract class, constructors in derived class.

4.1 INTRODUCTION:

- "The mechanism of deriving new class from an already existing class is called inheritance."
- The old class is referred to as the Base class and new class is referred to as the Derived class.
- The derived class inherits some or all features of the base class. Derived class can also add its own features.
- The Base class remains unchanged in this process..
- The main advantages of inheritance are the code reusability.
- The following figure shows the inheritance relation ship between base class and derived class.



- Direction of arrow indicates direction of inheritance.

ADVANTAGES OF INHERITANCE:

- Permits code Reusability:
  Once the base class is written and debugged, it needs not to be touched again. One can create a new class, which is derived from the base class. The newly derived class can inherit all the capabilities of base class and can also add its own features to the base class to work in different manner without making any changes to the base class.
- Reusing the existing code saves time and money and also increase program readability.
- Inheritance also helps in original conceptualization of a programming problem, and in the overall design of the program.

DEFINING DERIVED CLASS:

The general syntax of deriving a class is.

```
class derived-class-name : visibility-mode base-class-name
{
        // Body of derived class
};
```

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

- The visibility mode is optional. There can be two visibility modes

1) Private
2) Public

- The default visibility mode is private.
- The visibility mode specifies whether the features of base class are privately derived or publicly derived.

```
1)      class A : private AB     // private derivation
        {
        };
2)      class A : public AB              // Public derivation
        {
        };
```

```
3)      class A : AB            // By default private derivation
        {
        };
```

Ques : Explain three visibility modes (Access-specifiers) available in C++.

TYPES OF VISIBILITY MODES:

Access modifiers are also called as visibility mode:
There are three types of visibility modes available in c++.
The members of the class can be,

1) private
2) protected
3) public

Private:

Private members can be accessed only by the member functions of a class. They are not accessible outside the class.

Public:

Public members are accessible anywhere in the program.

Protected:

A member declared as 'protected' is accessed by member functions of its own class as well as member functions of the immediately derived class.

```
class A
{
   private :       // Default visibility –mode
                        // visible to member function of its own class
   protected :     // visible to member functions of own class as well as member //functions of derived class
   public :        // visible to all functions in the program.
};
```

Visibility – mode	Accessed from member functions of own class	Accessed from member fun of Derived class	Accessed from object of derived class
Public	YES	YES	YES
Protected	YES	YES	NO
Private	YES	NO	NO

NOTE :

- Member function of a class can access all type of data (private , protected , public)
- Private members are only accessed by member function of a class.
- Object of a class can access only public member of a class. They cannot access private or protected members of a class.
- Protected members are accessed only by member function of a class and member function of immediately derived class.

Deriving a class privately or publicly:

A class can be derived either privately or publicly from the base class.

- **Private derivation**

```
class B : private A
{
        //Body of derived class
};
```

- When a base class is privately inherited by a derived class ,'public members' of a base class becomes 'private members ' of the derived class.
- The result is, only the member functions of derived class can access public members of base class. Objects of derived class cannot access private members of base class.

- **Public derivation**

```
Class B : public A
{
        //Body of derived class
};
```

- When a base class is publicly inherited by a derived class ,'public members' of a base class becomes 'public members ' of the derived class.

- The result is both the member functions of derived class and objects of derived class can access public members of base class.

In both the cases private members of a base class are not inherited and therefore private members of a base class will never become a member of derived class.

The following examples explain the private and public derivation.

Example : Private derivation

```
class base
{
private :
        int a;     // a is a private member of class base , cannot be inherited
public :
         int b;  // b is public member of base , ready to be inherited
};
class  derived : private base         // private derivation  -  public members of base class
                                      // ( int b) becomes private member of derived class
{ private :   int c;
```

int b; → Inherited from base class, only member function of derived class can access

```
public:  int d ;                  / / public member of derived class , object can access it
        void  setdata ( int x ,int y )
        {
          b = x  ;          // ok
          c = y; // ok
        }
        void display()
      {
        cout << a;         // error : a is a private member of base class, cannot be inherited
        cout << b<<c<< d; //ok
      }
};
void main()
{
 derived d1;
 d1.setdata ( 10,20);      //OK  b =10 , c =20
 d1.d =40;         // will work b'cos d is a public member of class derived
 d1.b = 50 ; // error , now b is a private  member of derived, object cannot access
}
```

Example : public derivation

```
class base
{
private :
        int a;     // a is a private member of class a , cannot be inherited
public :
         int b;  // b is public member of class a – ready to be inherited
};
class  derived : public base          // public derivation  -  public members of base class
                                      // ( int b) becomes public member of derived class
{
private :
          int c;
```

int b; → Inherited from base class, member function of derived class and object of derived class can access it

```
public:    int d;
void  setdata ( int x ,int y )
        {
          b = x  ;          // ok
          c = y; // ok
```

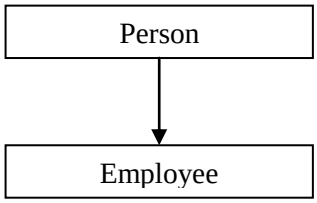
```
    }
    void display()
  {
    cout << a; //error
    cout << b <<c <<d; //ok
  }
};
void main()
{
 derived d1;
 d1.set ( 10,20);          //OK  b =10 , c =20
 d1.d =40;       // will work b'cos d is a public member of class derived
d1.b = 50 ; // ok , b is now public member of derived class , object can access it.
}
```

TYPES OF INHERITANCE

1) Single Inheritance
2) Multiple Inheritance
3) Hierarchical Inheritance
4) Multi-Level Inheritance
5) Hybrid Inheritance.

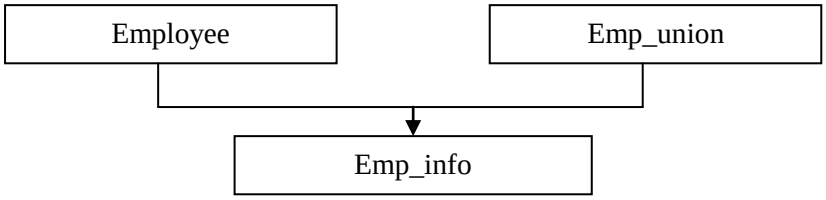
Single Inheritance:

The inheritance in which there is one base class and one derived class is called single inheritance.
(One base class and one derived class)



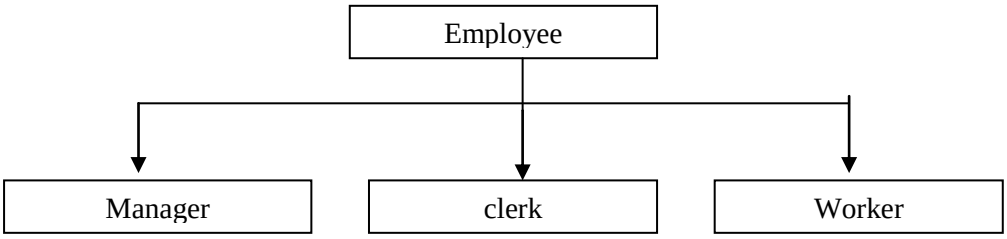
Multiple Inheritance:

Inheritance in which there is One derived class with several base classes is called multiple inheritance. (Several Base classes and only one derived class)



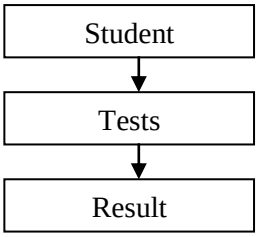
Hierarchical Inheritance

One base class may be inherited by several derived classes is called Hierarchical Inheritance.
(One base class several derived classes)



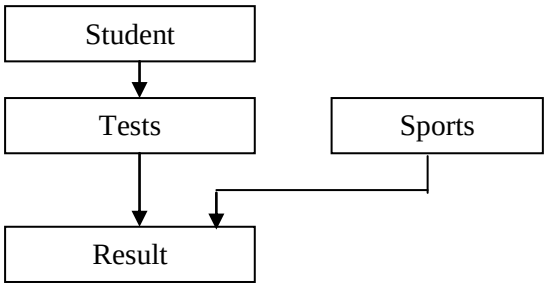
Multi-Level Inheritance:

A derived class can be derived from another derived class is called Multi level inheritance.



Hybrid Inheritance.

The mechanism of implementing more than one type of inheritances is called Hybrid inheritance.
For e.g. following diagram implements Multiple as well as multilevel inheritance.



❖ **SINGLE INHERITANCE (One base class and one derived class)**

```
Class base-class-name
{
// body of base class
};
class derived-class-name : visibility-mode  base-class-name
{
// body of derived class
};
```

Program base on single inheritance:

❖ Declare base class 'furniture' having data member's length, width and height. From that derive a class 'bookshelf' having data members as no f shelves.

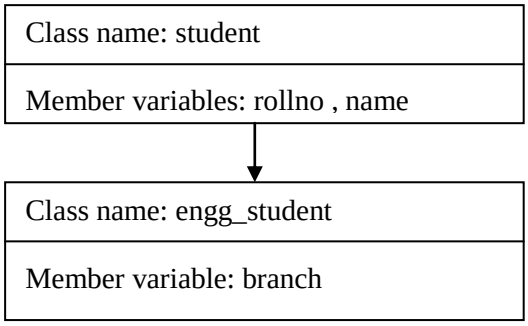
```
#include <iostream.h>
class furniture
{
  private: int l , w, h ;
  public: void  getdata (void)
            {      cout << "Enter length" ;
                   cin>>l;
                   cout <<"Enter width ";
                   cin>>w;
                   cout << "Enter height ";
                   cin >>h;
           }
           void showdata (void)
           {       cout << "length = " <<l <<endl;
                   cout << "Width = "<< w <<endl;
                   cout << "Height = "  << h <<endl ;
           }
};
class bookshelf : public furniture
{
            int no_shelf;
public:  void getdata (void)
           {       furniture :: getdata();
                   cout << "Enter no. of shelves ";
                   cin >> no_shelf;
           }
           void showdata (void)
           {       furniture :: showdata();
                   cout << "No. of shelves = "<< no_shelf;
           }
};
void main(void)
{
 bookshelf b1;
```

```
b1.getdata();
b1.showdata();
}
```

OUTPUT :

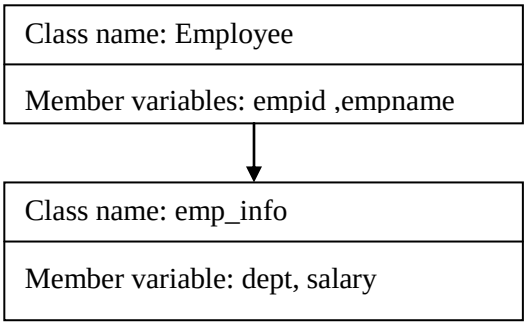
```
Enter length 70
Enter width 30
Enter height 10
Enter no of shelves 8
Length =70
Width = 30
Height =10
No. of shelves =8
```

❖ **Identify the type of inheritance and implement it by writing a program for the following figure**.



The type of inheritance is **single inheritance**

```
# include <iostream.h>
class student
{       int rollno;
        char  name [30];
public:  void getdata (void)
        {       cout << "Enter roll no " <<endl;
                 cin >> rollno;
                cout << "Enter name "<<endl;
                cin>>name;
        }
        void showdata (void)
        {       cout << "Rollno = " <<rollno <<endl;
                cout <<"Name = "<<name <<endl;
        }
};
class engg_student : public student
{
private :          char branch [20];
public: void getdata (void)
          {       student :: getdata();
                  cout << "enter branch " << endl;
                  cin >>branch;
           }
        void showdata (void)
        {       student :: showdata ();
                cout << "Branch = "<<branch<<endl;
        }
};
void main()
{
engg_student e1;
e1.getdata ();
e1.showdata();
}
```



The type of inheritance is **single inheritance**

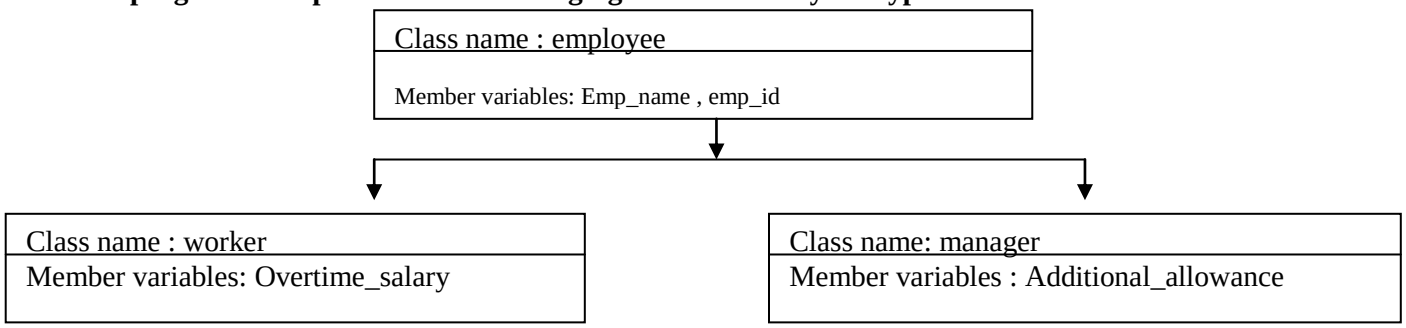
```
#include<iostream.h>
#include<conio.h>
class EMPLOYEE
{
private:
charemp_name[10] ;
intemp_id;
public:
void accept()
{
cout<<"\nentered the empid,_name”;
cin>>emp_id>>emp_name;
}
void display()
{
cout<<"\n emp name is:"<<emp_name;
cout<<"\n emp id is:"<<emp_id;
}
};
classEMP_info: public EMPLOYEE
{
private:
chardept[10] ;
fioat salary;
public:
void getdata()
{
cout<<"\nentered the dept name & salary of employee”;
cin>>dept>>salary;
}
void show()
{
cout<<"\n dept is:"<<dept;
cout<<"\n salary is:"<<salary;
}
};
void main()
{
EMP_INFO E1;
E1.accept();
E1.getdata();
E1.display();
E1.show();
getch();
}
```

❖ **Hierarchical Inheritance:**

Syntax of hierarchical Inheritance:

```
class A                    // base class
{
        // body of base class
};
class  B :  public A               // derived class 1
{
        // body of derived class 1
};
class  C : public A        // derived class 2
{
        // body of derived class 2
};
```

❖ **Write a program to implement the following figure also identify the type of inheritance.**



❖ Type of inheritance: **hierarchical**

```
# include <iostream.h>
class employee                    // base class
{
private : int empid
          char name [30];
public: void getdata(void)
        {       cout <<"Enter emp id: " <<endl;
                cin >> empid;
                cout <<"Enter emp name : "<<endl;
                cin >>name ;
        }
        void showdata(void)
        {       cout <<"emp id = " <<empid <<endl;
                cout <<"emp name = " << name << endl;
        }
};
class manager : public employee                    // manager is derived from class employee
{
private : float add_allowance;
public:  void getdata (void)
        {       employee :: getdata ();
                cout << "Enter additional allowance ";
                cin >>add_allowance;
        }
        void showdata (void)
        {       employee :: showdata ();
                cout << "Additional allowance = " << add_allowance;
        }
};
```

```
class worker : public employee  // worker is derived from class employee
{
private :float over_sal;
public:void getdata (void)
        {       employee :: getdata ();
                cout << "Enter overtime salary : "
                cin >>over_sal;
        }
        void showdata (void)
        {       employee :: showdata ();
                cout << "Over Time salary =   " << over_sal;
        }
};
void main ()
{
        manager m1 ; // create an object of manager
        cout << "Enter details of manager "<<endl;
        m1.getdata ();
        m1.showdata ();
        worker w1 ; // create an object of worker
        cout << "Enter details of worker "<<endl;
        w1.getdata();
        w1.showdata();
}
```

OUTPUT

```
Enter eetails of manager
Enter emp id :  123
Enter emp name : Dinesh
Enter additional allowance : 5000
Emp id = 123
Emp name = Dinesh
Additional allowance = 5000
Enter details of worker
Enter emp id : 234
Enter emp name : Nitin
Enter over time salary : 300
emp id = 234
emp name =  Nitin
over time salary = 300
```

Assignment:

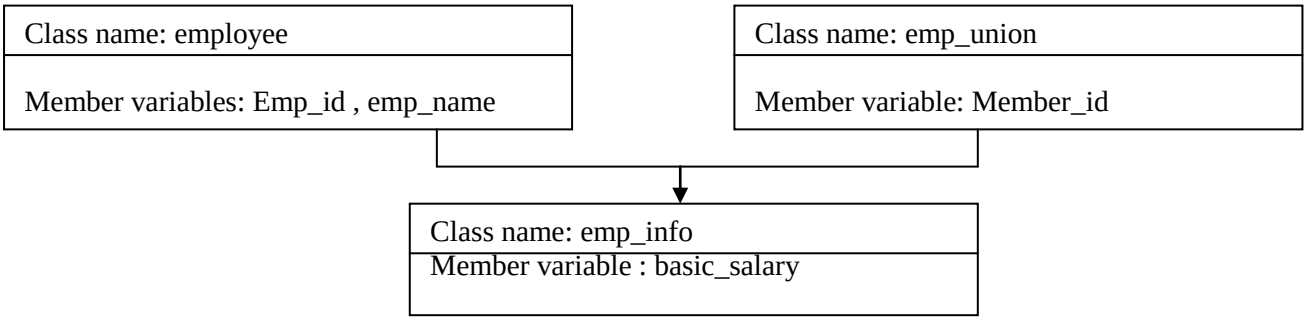
Create a class publication that stores the title and price. From this class ,derive two classes : book and audio_cassette. book will have 'author' as additional data member and audio_cassette will have 'playing_time' as additional data member. Each of these classes sholud have getdata() and showdata() functions to accept and display details .Write a program to implement the above inheritance.

Program based on Multiple inheritance: (several base classes and one derived class)

Syntax of multiple inheritance:

```
Class A // base class
{
        // body of base class A
};
class B  // base class
{
        // Body of base class B
};
class c  : public A , public B                  // class c is derived from class A and  B
{
        // Body of derived class C
};
```

❖ **Study the following diagram and identify the type of inheritance. Write class definition for each class with suitable member functions to accept and display data.**



❖ Ans : Type of Inheritance is Multiple Inheritance

```
# include <iostream.h>
class employee                    // base class
{
private : int emp_id
          char emp_name [30];
public: void getdata(void)
          {         cout <<"Enter emp_id : " <<endl;
                    cin >> emp_id;
                    cout <<"Enter emp_name : "<<endl;
                    cin >> emp_name ;
          }
          void showdata(void)
          {         cout <<"emp_id = " <<emp_id <<endl;
                    cout <<"emp_name = " << emp_name << endl;
          }
};
class emp_union                           // base class
{
private :         int member_id
public:void getdata(void)
          {
          cout <<"Enter Member id: " <<endl;
          cin >> member_id;
          }
          void showdata(void)
          {
          cout <<"Member id = " << member_id <<endl;
          }
};
class emp_info : public employee , public emp_union  // derived class
{
private :         float basic_sal;
public: void getdata(void)
          {         employee :: getdata();
                    emp_union :: getdata();
                    cout <<"Enter Basic Salary : "<<endl;
                    cin >> basic_sal ;
          }
          void showdata(void)
          {         employee :: showdata();
                    emp_union :: showdata();
                    cout <<" Basic Salary  = " << basic_sal <<endl;
          }
};
void main()
{
```

```
 emp_info e1;
 e1.getdata();
cout << "Details of Employee"<<endl;
 e1.showdata();
}
```

OUTPUT

```
Enter emp_id : 1234
Enter emp_name : deepali
Enter Member_id : 4567
Enter Basic Salary :8000
Details of Employee
emp_id = 1234
emp_name = deepali
Member_id = 4567
Basic Salary = 8000
```

- **Ambiguity in Multiple Inheritance:**

Consider the following example. What will be the output?

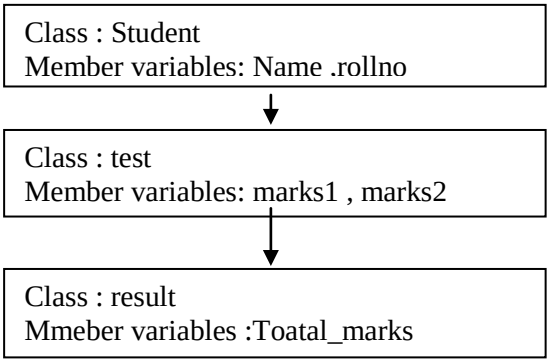
```
Class A            // Base class
{     public: void show()
                  {
                   cout << "In class A ";
                   }
};
Class B            // Base class
{     public:      void show()
                   {
                    cout << "In class B ";
                   }
};
Class  C : public A ,public B                // class C is derived from A and B
{
};

int main()
{
 C obj;
 Obj.show();      // ambiguity error: will not work
Obj . A::show() ;          // OK Invokes show() of class A
Obj . B::show() ;          // OK Invokes show() of class B
return 0;
}
```

In the above program the statement, obj.show();
Will report ambiguity error this is because class C has two base classes- class A and class B. And both the base classes have the same function show (). The derived class C does not override the function show (). So compiler tries to invoke the base class function, but here there are two versions of base class function 1) A**::** show() 2) B**::**show(). Compiler cannot make decision which version is to call.

Program based on Multilevel Inheritance Syntax : Multilevel Inheritance

```
class A            // Base class
{       // Body of base class
};
class B : public A                  // B is derived from A
{    // body of class B
};
class C : public B          // C is Derived from B
{  // Body of class C
};
```

❖ **Write a program to implement following inheritance.**




```
# include <iostream.h>
class student
{
private : int rollno;
          char name[30];
public: void getdata(void)
          {         cout <<"Enter rollno: " << endl;
                    cin>>rollno;
                    cout << "Enter name : "<<endl
                    cin >> name;
          }
                    void showdata(void)
          {         cout <<"Rollno= " <<rollno << endl;
                    cout << "Name = "<< name << endl;
          }
};
class test  : public student
{
protected :       int marks1 , marks2;      // merks1 and marks2 are protected members
public :void getmarks ()
          {         cout << "Enter marks of sub1 : " ;
                    cin >> marks1;
                    cout << "Enter marks of sub2 : " ;
                    cin >> marks2;
          }
          void showmarks (void)
          {
                    cout << "Marks of sub1 = " << marks1;
                    cout << "Marks of sub2 = "<< marks2;
          }
};
class result :  public test
{
private :         int total_marks;
public:void display_result()
          {
                    total_marks = marks1 + marks2;          // marks1 and marks2 are accessible
                                                            // here b'cos they are declared as
                                                            // protected in base class test
                    student :: showdata();
                    test :: showmarks () ;
                    cout << "Total Marks = "<< total_marks;
        }
};
void main()
{
result r1;
```

```
r1.getdata();                  // get rollno and name
r1.getmarks();    // get marks1 and marks2
cout << “details of student: “ <<endl;
r1.display_result();
}
```

OUTPUT

```
Enter Rollno : 123
Enter Name : Deepali
Enter Marks of sub1: 45
Enter Marks of sub2 : 38
Details of student :
RollNo = 123
Name = Deepali
Marks of sub1 : 45
Marks of sub2
```

Program based on Hybrid Inheritance Syntax : Hybrid Inheritance

```
class A              // Base class
{         // Body of base class
};
class B : public A                    // B is derived from A
{    // body of class B
};
class C                  // base class
{  // Body of base class C
};
class D : public B, public C        // D is Derived from B and C
{  // Body of class D
};
```

```
#include <iostream.h>
#include <conio.h>
class student
{
protected:
int rno;
public: void getnum()
{
cout<<"enter the rno:";
cin>>rno;
}
void putnum()
{
cout<<rno"\n";
}
};
class test: public student
{
protected: int m1,m2;
public: void getmarks()
{
getnum();
cout<<"enter the mark1:";
cin>>m1;
cout<<"enter the mark2:";
cin>>m2;
}
```

```
void putmarks()
{
putnum();
cout<<m1"\t";
cout<<m2"\t";
}
};
class sports
{
protected:
float score;
public:
 void getscore()
 {
 cout<<"enter the score value:";
 cin>>score;
 }
 void putscore()
 {
 cout<<score"\t";
 }
};
class result: public test,public sports
{
protected:
float total;
public:
 void display()
 {
 total=m1+m2+score;
 putmarks();
 putscore();
 cout<<"total marks = <<total"\t"< }
};
int main()
{
clrscr();
result r1;
r1.getmarks();
r1.getscore();
cout<<"details of student marks are:"<<endl;
r1.display();
getch();
return 0;
}
```

OUPUT:

```
Enter the rno:1
Enter the mark1:56
Enter the mark2:89
Enter the score value:58
details of students marks are:
1 56 89 58 203
```

CONSTRUCTORS IN MULTIPLE INHERITANCE

If any base class contains constructor with one or more arguments, then it is mandatory for the derived class to have a constructor and pass arguments to the base class constructor:

The general form of defining derived class constructor is,

- The constructor of the derived class receives the entire list of values and passes them to the base class constructors in the order in which they are declared in the derived class.

- Constructors are always called in order from the Base class to the most derived class constructor. That is the base class constructor is called first and then the derived class constructor is called.
- This is because derived class can access any public or protected data of the base class. Hence by the time they are accessed, they must be initialized. This can be assured by invoking base class constructor before the derived class constructor.

```
Derived-class-name( arg1 , arg2 , arg3, arg4,…….,arg N) :
base1(arg1 ,arg2) , base2( arg3 ,arg4) , ……..baseN (arg N)
{ // body of the derived constructor
}
```

Example:

```
Class A
{       int a1 , a2;
public:  A (int x  ,int y)              // constructor
        {
                a1 = x ;
               a2 =y;
        }
};
Class B
{       int b1 , b2;
public: B (int x ,int y)    // constructor
        { b1 = x ;
          b2 =y;
        }
};
class D : public A ,public B
{  int d1,d2;
public : D ( int k1 ,int k2 ,int k3 ,int k4 , int k5 ,int k6) : A ( k1,k2) , B (k3,k4)  // constructor
```

call const. of A call const. of B

```
{
   d1 = k5 ;
   d2 =k6;
}
};
```

Execution of Base class constructors:

Method of Inheritance	**Order of execution**

1) Class B : public A
{
};
	A() ; base class cont.
	B() ; Derived class const.

2) class A : punlic B ,public C
{
}
	B() ; base class const
	C() ; Base class const
	A() ; derived class const

3) class A : public B , virtual public C
{
};
	C() ; virtual base class const
	B() ; ordinary base class const
	A() : derived calss const

4)
class A : private B , virtual public C , virtual public D
{
};
	C() ; virtual base class const
	D (); virtual base class const
	B() : ordinary base class const
	A() ; Derived class const.

- The constructor for virtual base class is invoked before the non- virtual base classes. If there are more than one virtual base class them they are invoked in the order in which they are declared.

NOTE: Destructors are called in reverse order that is the most derived class destructor is invoked first and works towards base class.

ABSTRACT CLASSES:

- An abstract class is one that is not used to create any object. An abstract class is designed only to act as a base class which can be inherited by other classes.
- Consider the following example

```
class employee
{
};
class manager :public employee
{
};
class scientist :public employee
{
};
class laborer :public employee
{
};
```

- In the above example employee acts as an abstract class. It is not used to create objects; it acts as a base class and is inherited by manager, scientist, and laborer classes.
- One way to make any class abstract is to place at least one pure virtual function in a base class. Now if you try to create an object of that class, compiler will report an error.

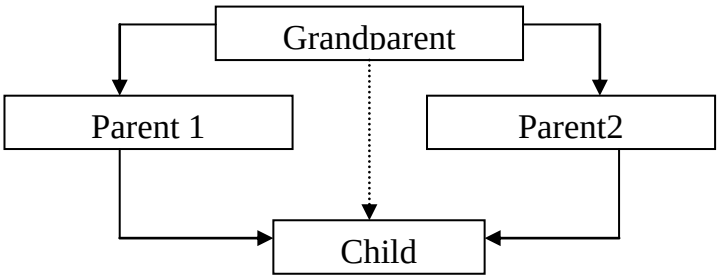
```
class employee            // abstract  class
{
public:
virtual void display()= 0;          // pure virtual fucntion
};
```

NOTE : pure virtual function is a function with no body .

VIRTUAL BASE CALSS:

The duplication of inherited members due to these multiple paths can be avoided by making the common base class (ancestor class) as virtual base class while declaring the direct or intermediate base class.
Consider a situation where all three kinds of inheritance multiple, multilevel and hierarchical are involved.

- Here in the above diagram the child has two direct base classes ‘ parent1’ and ‘parent2’ which themselves have a common base class ‘grandparent’.



- The ‘child’ class inherits the traits of ‘grandparent’ via two separate paths. All the public and protected members of ‘grandparent’ are inherited twice via ‘parent1’ and again via ‘parent2’.
- This means ‘child’ class would have duplicate sets of members inherited from ‘grandparent’ this introduces ambiguity& should be avoided.
- To avoid duplication of inherited members due to the multiple paths we use virtual base classes.
- Consider the following example,

```
# include <iostream.h>
class A          // grandparent
{
};
class B1 :  virtual public A              // parent1
{
};
class B2 : virtual public A               //parent2
```

```
{
}
class C : public B1 , public B2   // child
{
// only one copy of a will be inherited
};
```

When a class is made a virtual base class,C++ compiler takes necessary care to see that only one copy of that class is inherited.

MEMBER CLASSES: (Nesting of classes)

- An object can be a collection of many other objects. That is a class can contain objects of other classes as its data member.

Example:

```
Class alpha { ….};
Class beta {….};
Class gamma              // gamma act as a container class
{
        alpha a;// a is an object of alpha class
        beta b;          // b is an object of beta class
};
```

All objects of gamma class contains the object a and b. This kind of relationship is called containership or nesting. Here class gamma is called container class.

Ques : What are the implications of the following definitions? (Dec-2003)

1) class A : public B , public C { …};
2) class B : private C , public A {….};

Ans

1) In the first case class A is derived publicly from class B and class C. The result is, all the public members of class B and C become the public members of class A. Hence all public members of base class B and C can be accessed by the object of derived class C.
2) In the second case class B is derived privately from class C and publicly from class A.
   - Because the class B is inherited using the public keyword , all public members of class A becomes the public member of class B and therefore are available to the object of derived class B.
   - But since class B is inherited using the private visibility mode, all public members of the class C becomes private members of class B. Therefore object of class B cannot access public members of base class C directly.