SATHYABAMA UNIVERSITY FACULTY OF ELECTRICAL AND ELECTRONICS

OBJECT ORIENTED PROGRAMMING (SCS1202)

UNIT 4

INHERITANCE

Base class and derived class relationship-Derived class declaration-Forms of inheritance-Inheritance and member accessibility- Constructors in derived class-Destructors in derived class-Multiple inheritance-Multi level inheritance-Hybrid inheritance-Virtual base classes-Member function overriding-Virtual functions-Abstract classesPure Virtual functions.

INHERITANCE

The mechanism of deriving a new class from an old one is called inheritance(or derivation). The old class is referred to as the base class and the new one is called the derived class. The derived class with only one base class is called single inheritance and one with several base classes is called multiple inheritance. On the other hand, the traits of one class may be inherited by more than one class. This process is known as hierarchical inheritance. The mechanism of deriving a class from another 'derived class' is known as multilevel inheritance. The following figure shows various forms of inheritance that could be used for writing extensible programs. The direction of arrow indicates the direction of inheritance.

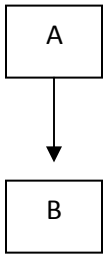
DEFINITION:

The mechanism of deriving the new class from an old one is called inheritance. The old class is called as base class and the new class is called as derived class.

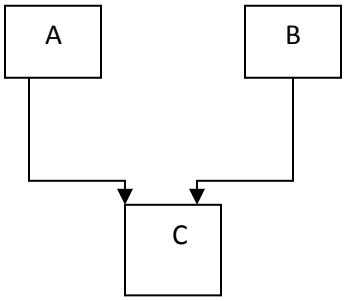
Types of Inheritance:

1. Single inheritance
2. Multiple inheritance
3. Multilevel inheritance
4. Hierarchical inheritance(refer Assignment)
5. Hybrid inheritance

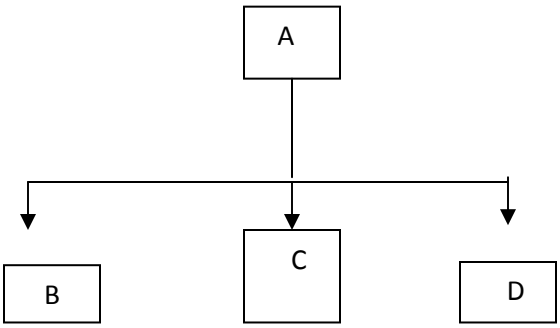
Single Inheritance



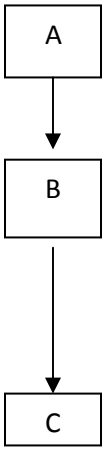
Multiple Inheritance



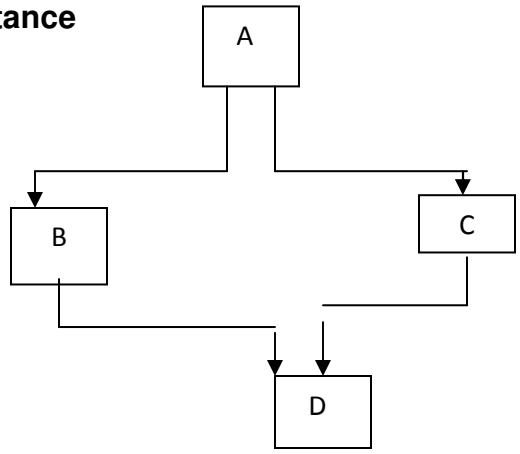
Hierarchical Inheritance



Multilevel Inheritance



Hybrid Inheritance



Defining derived classes:-

A derived class is defined by specifying its relationship with the base class in addition to its own details.

The general form:-

class derived-class-name: visibly-mode base-class-name

{

----------//

----------//members of derived class

};

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

Examples:-

class ABC:private XYZ//private derivation

{

```
        members of ABC
};
class ABC:public XYZ//public derivation
{
        members of ABC
};
class ABC:XYZ//private derivation by default
{
        members of ABC
};
```

When base class is privately inherited by a derived class, 'public members' of the base class become 'private members' of the derived class and therefore the public members of the base class can only be accessed by the member functions of the derived class. They are inaccessible to the objects of the derived class. Remember, a public member of a class be accessed by its own objects using the dot operator. The result is that no member of the base class is accessible to the objects of the derived class.

When the base class is publicly inherited, 'public members' of the base class become 'public members' of the derived class and therefore they are accessible to the objects of the derived class. In both the cases, the private members are not inherited and therefore, the private members of the base class will never become the members of its derived class.

1.SINGLE INHERITANCE:-

The new class can be derived from only one base class is called single inheritance.

Let us consider a simple example to illustrate inheritance. The following program shows the base class B and a derived class D. The class B contains one private data

member, one public data member, and three public member functions. The class D contains one private data member and two public member functions.

```
#include<iostream.h>
class B
{
        int a;   //private; not inheritable
        public:
                int b;   //public; ready for inheritance
                void get_ab()
                {
                        a=5;
                        b=10;
                }
                int get_a(void)
                {
                        return a;
                }
                void show_a(void)
                {
                        cout<<"a="<<a<<endl;
                }
};
class D:public B      //public derivation
{
        int c;
        public:
                void mul(void)
                {
                        c=b*get_ab();
```

```
        }
        void display()
        {
                cout<<"a="<<get_ab()<<endl;
                cout<<"b="<<b<<endl;
                cout<<"c="<<c<<endl;
        }
};
void main()
{
        D d;
        d.get_ab();
        d.mul();
        d.show_a();
        d.display();
        d.b=20;
        d.mul();
        d.display();
}
```

Output:-

```
a=5
a=5
b=10
c=50

a=5
b=20
c=100
```

The class D is a public derivation of the class B. Therefore, D inherits all the public members of B and retains their visibility. Thus a public member of the base class B is also a public member of the derived class D. The private members of B cannot be

inherited by D. The program illustrates that the objects of class D have access to all the public members of B. Let us have a look at the functions show_a() and mul().

```
void show_a()
{
        cout<<"a="<<a<<endl;
}
void mul()
{
        c=b*get_a();//c=b*a
```

Although the data member a is private in B and cannot be inherited, objects of D are able to access it through an inherited member function of B.

Let us now consider the case of private derivation.

```
        #include<iostream.h>
class B
{
        int a;
        public:
                int b;
                void get_ab();
                int get_a(void);
                void show_a(void);
};
class D:private B
{
        int c;
        public:
                void mul(void);
                void display();
};
```

In private derivation, the public members of the base class become private members of derived class. Therefore, the objects of D can not have direct access to the public member functions of B.The statement such as d.get_ab(),d.get_a(),d.show_a() will not work.

Program:-

```
#include<iostream.h>
class B
{
        int a;//private; not inheritable
        public:
                int b;//public; ready for inheritance
                void get_ab()
                {
                        a=5;
                        b=10;
                }
                int get_a(void)
                {
                        return a;
                }
                void show_a(void)
                {
                        cout<<"a="<<a<<endl;
                }
};
class D:private B//private derivation
{
        int c;
        public:
                void mul(void)
                {
```

```
                    c=b*get_ab();
                }
                void display()
                {
                    cout<<"a="<<get_ab()<<endl;
                    cout<<"b="<<b<<endl;
                    cout<<"c="<<c<<endl;
                }
};
void main()
{
    D d;
    // d.get_ab(); it won't work
    d.mul();
    //d.show_a(); won't work
    d.display();
    //d.b=20; won't work
    d.mul();
    d.display();
}
```

Output:-

```
a=5
b=10
c=50

a=12
b=20
c=240
```

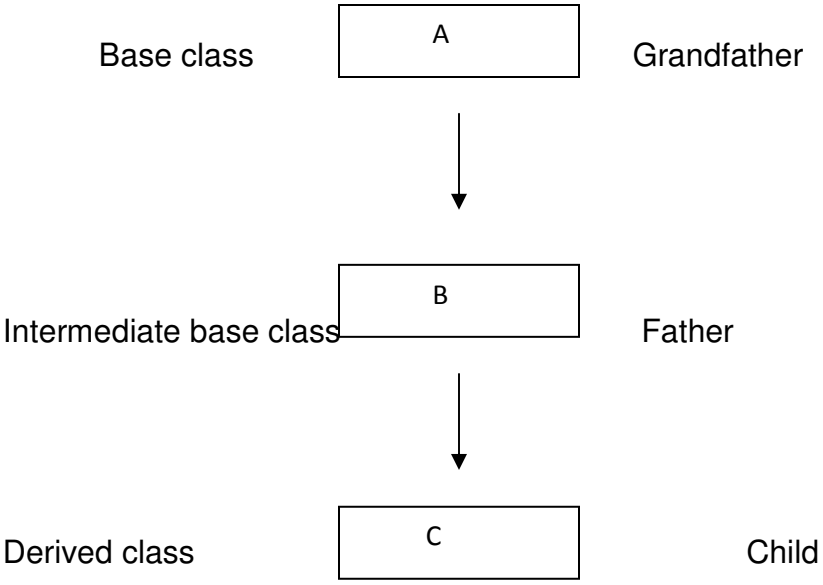
A private member of a base class cannot be inherited and therefore it is not available for derived class directly. A protected which serve a limited purpose in inheritance. A member declared as protected is accessible by the member functions within its class and any class immediately derived from it. It cannot be accessed by the functions outside these two classes. A class can now use all the three visibility modes as illustrated below:

```
class alpha
{
        private:
                ------- //optional
                ------- // visible to member functions
                ------- // within its class
        protected:
                ---------- // visible to member functions
                ---------- // of its own and derived class
        public:
                ---------- // visible to all functions
                ---------- //in the program
}
```

When a protected member is inherited in public mode, it becomes protected in the derived class too, and therefore is accessible by the member functions of the derived class. It is also ready for further inheritance. A protected member, inherited in the private mode derivation, becomes private in the derived class. Although it is available to the member functions of the derived class, it is not available for further inheritance.

2.Multilevel Inheritance:-

The class **A** serves as a base class for the derived class **B** which in turn serves as a base class for the derived class **C.** The chain **ABC** is known as inheritance path.



A derived class with multilevel inheritance is declared as follows.

class A(….) ;

class B:public A(…..);

class C:public B(…..);

The multilevel inheritance is defined as, the new class is derived from another derived class.

Here the class A serves as a base class for the derived class B which in turn serves as a base class for the derived class C. The derived class is defined as,

```
Class A
{
        //body of base class A
};
Class B : public A
{
        //body of intermediate base class B
};
```

Class C : public B

{

//**body of derived class**

};

Example Program

EXAMPLE :1

```
#include <iostream.h>
class student
{
        protected:
                int rollno:
        public:
                void getroll(int x)
                {
                rollno = x;
                }
};
void test: public student
{
        protected:
                float sub1, sub2;
        public:
                void getmart(int y, int z)
                {
                sub1 = y; sub2= z;
                }
};
void result : public test
{
        float total;
```

```
        public:
        void display( )                 {
total =  sub1 + sub2;
cout<<rollno<<sub1<<sub2;
cout<<total;
        }
};
void main( )
{
        result s;
        s.getroll(101);
        s.getmark(90, 100);
        s.display( );
```

EXAMPLE :2

```
#include<iostream.h>
class student
{
        protected:
                int roll_number;
        public:
                void get_number(int);
                {
                        roll_number=a;
                }
                void put_number(void)
                {
                        cout<<"Roll Number:"<<roll_number<<endl;
                }
};
class test: public student  //First level derivation
{
```

```
        protected:
                float sub1;
                float sub2;
        public:
                void get_marks(float,float)
                {
                        sub1=x;
                        sub2=y;
                }
                void put_marks(void)
                {
                        cout<<"Marks in sub1="<<sub1<<endl;
                        cout<<"Marks in sub2="<<sub2<<endl;
                }
};
class result : public test // second level derivation
{
        float total;
        public:
                void display(void)
                {
                        total=sub1+sub2;
                        put_number();
                        put_marks();
                        cout<<"Total="<<total<<"\n";
                }
);
void main()
{
        result student1;
        student1.get_number(111);
```

```
        student1.get_marks(75.0,59.5);
        student1.display();
}
```

Output:-

```
        Roll Number:111
        Marks in sub1:75
        Marks in sub2:59.5
        Total:134.5
```

3.MULTIPLE INHERITANCE:-

A class can inherit the attributes or properties of two or more classes that is a new class can be derived from more than one base class is called multiple inheritance. Here the class A , class B and class C serves as a base class for the derived class D

The derived class definition syntax is:

```
class d: visibility base-1, visibility base 2,…
{
        body of class D;
};
```

Example:

```
#include<iostream.h>
class A
{
 protected:
        int rollno;
public:
        void getroll(int x)
        {
```

```
        rollno=x;
        }
};
class B
{
protected:
        int sub1,sub2;
public:
        void getmark(int y,int z)\
        {
        sub1=y;
        sub2=z;
        }
};
class C : public A, public B
{
        int total;
        public:
        void display()
        {
        total=sub1+sub2;
        cout<<"roll no:"<<rollno<<"sub1:"<<sub1<<"sub2:"<<sub2;
        cout<<"total"<<total;
        }
};
void main()
{
        C s;
        s. getroll(435);
        s.getmark(100,90);
        s.display();
```

```
}
```

Run:

```
Roll no : 435
Sub1: 100
Sub2: 90
```

4.HIERARCHICAL INHERITANCE:-

The hierarchical inheritance structure is given below .This type is helpful when we have class hierarchies. In this scheme the base class will include all the features that are common to the subclasses.

```
#include<iostream.h>
#include<string.h>
class A
{
protected:
int x, y;
public:
void get ( )
{
cout<<"Enter two values"<<endl;
cin>> x>>y;
}
};
class B : public A
{
private:
int m;
public:
void add( )
{
m= x + y;
cout<<"The Sum is "<<m;
```

```
}
};
class C : public A
{
private:
int  n;
public:
void mul( )
{
n= x * y;
cout << “The Product is “<<n;
}
};
class D : public A
{
private:
float  l;
public:
void division( )
{
l = x / y;
cout <<”The Quotient is “<< l;
}
};
void main( )
{
B obj1;
C obj2;
D obj3;
B .get( );
B .add( );
C .get( );
C .mul( );
```

```
D .get( );
D .division( );
}
```

Output of the Program

```
Enter two values
    12 6
The Sum is 18
The Product is 72
The Quotient is 2.0
```

5.HYBRID INHERITANCE:-

Hybrid inheritance = multiple inheritance + multi-level inheritance

The hybrid inheritance is defined as a combination of multilevel and multiple inheritances. This new class can be derived by either multilevel or multiple method or both.

Example program

In this program the derived class (result) object will be inheriting the properties of both test class and sports class.

```
#include <iostream.h>
class student
{
        protected:
                int rollno;
        public:
                void getroll (int x)
                {
                rollno = x;
}
};
class test: public student
{
```

```
protected:
        int sub1, sub2'
public:
        void getmark (int y, int z)
{
        sub1 = y;  sub2 = z;
}
};
class  sports
{
protected:
        int score;
public:
        void  getscore (int a )
        {
        score=a;
        }
};
class result: public test, public sports
{
        int total;
public:
        void display()
        {
        total= sub1+sub2+score;
        cout<<"rollno:"<<rollno<< "total:"<<"Score:"<<total;
        }
};
void main( )
{
        result S;
```

```
        s.getroll(101);
        s.getmark(90,98);
        s.getscore(2000);
        s. display( );
}
```

Run:

Rollno: 101

Total: 188

Score: 2000

VIRTUAL FUNCTION

When we use the same function name in base and derived classes, the function in the base classes is declared as virtual using keyword 'virtual' preceding its normal declaration.

The member function that can be changed at runtime is called virtual function.

```
Class classname
{
        public:
        . . . . .
        virtual returntype functionname (arguments)
        {
                . . . . .
                . . . . .}};
```

The virtual function should be defined in the public section of a class. When one function is made virtual and another one is normal, and compiler determines which function to call at runtime. In this we have to create a 'basepointeer'.

If the basepointer paints to a base class object means it will execute the baseclass function.

If the basepointer points to a derived class object means it will execute the derived class function.

Example Program:

```
#include <iostream.h>
Class A
{
        public:
        void displayA( )
        {
                cout<<”Welcome”;
        }
        virtual void show( )
        {
                cout<<”Hello”;
        }
};
class B: public A
{
        public:
        void display( )
        {
                cout<<”Good Morning”;
        }
        void show( )
        {
                cout<<”Hai”;
        }
};
void main( )
{
        A S1;
```

```
        B S2;
        A *bptr;
        bptr = &S1;
        bptr→show( );              //Calls base class show
        bptr=&S2( );
        bptr→show( );              //Calls derived class show
        S2. displayA( );     //Calls base class displayA
        S2.display( );       //Calls derived class display
}
RUN:
Hello
Hai
Welcome
Good Morning
```

Explanation:

The base pointer first holds the address of the base class object means it will run the base class member function display and show will be executed. Next the base pointer points to the derived class object, in this before executing the derived class function if will check whether the corresponding base class function is 'virtual' or not, if it is not virtual then execute the base class function otherwise it will execute the derived class function.

Rules for virtual function:

1. Virtual function must be a members of some class.
2. They cannot be static members.
3. They are accessed by using base pointer.
4. A virtual function in a base class must be defined eventhough it may not be used.
5. A prototype (or declaration) of a base class virtual function and all the derived class version must be identical.
6. We cannot we a pointer to a derived class to access an object of the base class.

VIRTUAL BASE CLASS:-

The duplication of inherited members due to the multiple paths can be avoided by making the common base class as virtual base class while declaring the direct or intermediate base classes as shown below.

```
class  A
{
        ------
        ------
};
class B1:virtual public A
{
        -----
        -----
};
class B2:public virtual A
{
        -----
        -----
};
class C:public B1,public B2
{
        -----
        -----
};
```

When a class is made a virtual base class, C++ takes necessary care to see that only one copy of that class is inherited, regardless of how many inheritance paths exist between the virtual base class and a derived class.

```
class A
{
        private:
```

```
        int a;
    public:
        void funcA()
        {
            cin>>a;
            cout<<"a="<<a;
        }
};
class B:virtual public A
{
    private:
        int b;
    public:
        void funcB()
        {
            cin>>b;
            cout<<"b=">>b;
        }
};
class C:public virtual A
{
    private:
        int c;
    public:
        void funC()
        {
            cin>>c;
            cout<<"c=">>c;
        }
};
class D : public B, public C
```

```
{
        private:
                int d;
        public:
                void funD()
                {
                        cin>>d;
                        cout<<"d=">>d;
                }
};
void main()
{
        D d1;
        C c1;
        c1.funC();
        d1.funA();
        d1.funB();
        d1.funC();
        d1.funD();
}
```

Output:-

5
c=5
4
a=4
6
b=6
8
c=8
9
d=9

Abstract classesPure Virtual functions.

The virtual function doesn't have any operation is called "do-nothing" function or pure virtual function.

This is represented as

virtual void display()=0;

Such functions are called pure functions. This pure virtual class is also called abstract base class.