Unit-5

Inheritance

Date.
Page No. 10

Inheritance is the process by which the objects of one class acquire the properties of objects of another class. It helps to share common characteristics with the class from which it is derived.

For example:- The bird 'Robin' is a part of class 'flying bird' which is again a part of class 'bird'.

The concept of inheritance provides the idea of reusability. This means that we can add additional features to an existing class without modifying it.

In inheritance the old class from which features are transfered to new class is known as base class or parent class while the new classes that inherit some or all properties of base class are known as derived or child class. There are different types of inheritances based on no. of base classes with respect to derived classes as follows:-

i) Single inheritance
ii) Multiple inheritance
iii) Multilevel inheritance
iv) Hierarchial inheritance
v) Hybrid inheritance.

* Defining Derived Classes:-

A derived class can be defined by specifying it's relationship with the base class in addition to

its own details. The general form of defining a derived class is:-

(public, private or protected)

(Keyword)

```
class derived-class-name:visibility-mode base-class-name
{
.....// members of derived class
.....// i.e, data members & member functions
.....//
};
```

Examples:

i) class ABC: private XYZ //private definition.
{
members of ABC
};

ii) class ABC: public XYZ // public definition.
{
members of ABC
};

Note: (i) When the base class is publicly inheritated, 'public members' of the base class becomes 'public members' of the derived class and therefore they are accessible to the objects of the derived class.

(ii) When a base class is privately inheritated by a derived class, 'public members' of the base class becomes '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 objects of the derived class. Remember, a public

(i), (ii), (iii) Conclusion

private → visible to member function of it's own class only.
protected → visible to member functions of it's own and derived class.
public → visible to all functions in the program.

member of a class can be accessed by it's own objects using the dot operator. The result is that no ~~matter~~ member of the base class is accessible to the objects of the derived class.

(iii) When a base class is inherited with protected by derived class then it can be accessed within class and from any class derived from this class. These members can't be accessed from outside of these (same as in derived class) and hence help to achieve the concept of data hiding.

Overriding member functions:

We can define data member and member function with the same name in both base and derived class. When the function with same name exists in both class and derived class, the function in the derived class will get executed. This means the derived class function overrides the base class function.

Example

```
Class baseA{
Public:
    void getdata(){
    :::
    }
};

Class derived B: public baseA{
Public:
    void getdata(){
    :::
    }
};

Void main(){
    Derived B obj;
    obj.getdata();
    getdata();
}
```

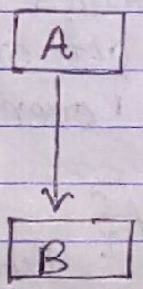
When the statement obj.getdata(); gets executed, the function getdata() of the derived class i.e, of derived B get executed. This means, the derived class function overrides the base class function.

The scope resolution (::) operator can be used to access base class function through an object of the derived class.

⊛ Types of Inheritance:-

1) Single inheritance:

The derived class with only one base class is called single inheritance.



Single inheritance

Program to understand single inheritance: (Publicly)

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

class B{
    int a; // private; not inheritable
public:
    int b; // public; ready for inheritance
    void get_ab();
    int get_a();
    void show_a();
};
```

derived class name

base class with public specifier

```
class D: public B      //public derivation
{
    int c;
public:
    void mul();
    void display();
};

void B::get_ab(){
        a=5; b=10;
}

int B::get_a(){
        return a;
}

void B::show_a(){
        cout<<"a="<<a<<"\n";
}

void D::mul(){
        c=b*get_a();
}

void D::display(){
        cout<<"a="<<get_a()<<"\n";
        cout<<"b="<<b<<"\n";
        cout<<"c="<<c<<"\n\n";
}

int main(){
    D d;
    d.get_ab();
    d.mul();
    d.show_a();
    d.display();
    d.b=20;
    d.mul();
    d.display();
    return 0;
}
```

Object is always created with the help of derived class not by base class and it helps to access data

Output

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

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

In this program 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 a base class B is also a public member of the derived class D. The private members of B can not be inherited by D.

Program to understand single inheritance: (Privately).

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

class B {
        int a; //private; not inheritable
    public:
        int b; //public; ready for inheritance.
    void get_ab();
    int get_a();
    void show_a();
};

class D; private B    //private derivation.
{
    int c;
    public: void mul();
            void display();
};
void B::get_ab(){
        cout<<"Enter values for a and b";
        cin>>a>>b;
    }
int B::get_a(){
        return a;
    }
```

derived class name

Base class with private specifier

```
void B::show_a(){
        cout<<"a="<<a<<"\n";
    }

void D::mul(){
        get_ab();
        c=b*get_a();          //'a' cannot be used directly for private
    }

void D::display(){
        show_a(); //outputs value of 'a'
        cout<<"b="<<b<<endl;
        cout<<"c="<<c<<endl;
    }

int main(){
        D d;
        // d.get_ab(); // Won't work
        d.mul();
        // d.show_a(); // Won't work
        d.display();

        // d.b=20; Won't work; b has become private
        d.mul();
        d.display();
        return 0;
    }
```

(for understanding data access in private)

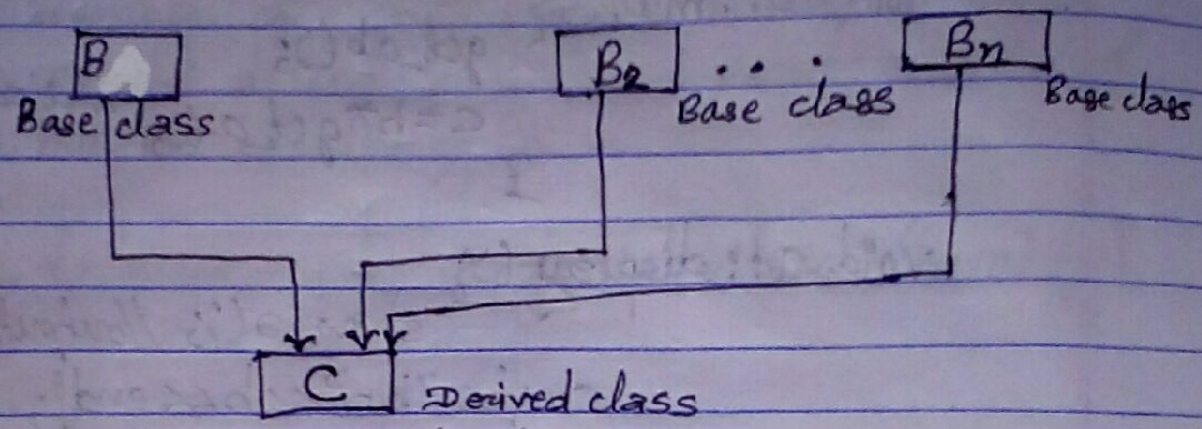
Output

```
Enter values for a and b
a=5
b=10
c=50
Enter value for a and b
a=12
b=20
c=240
```

Date.
Page No.

2) Multiple inheritance:

The derived class with more than one base class is called multiple inheritance.



Multiple inheritance

The syntax of a derived class with multiple base class is as follows:-

```
class D: visibility B1, visibility B2, ...
{
    ::: (Body of D)
};
```

where visibility may either be public or private, or protected.

Program to understand multiple inheritance: (Protected).

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

class M{
protected:
    int m;
public:
    void get_m(int);
};
```

```
class N{ protected:
            int ;
         public:
            void get_n(int);
       };

class P: public M, public N {
                    public:
                        void display (void);
                        };

void M::get_m(int x) {
                        m=x;
                      }

Void N:: get_n (int y) {
                        n=y;
                       }

void P:: display (void){
                        cout<<"m="<<m<<"\n";
                        cout<<"n="<<n<<"\n";
                        cout<<"m*n="<<m*n<<"\n";
                       }

int main (){
            P p;
            p.get_m(10);
            p.get_n(20);
            p.display ();
            return 0;
           }
```

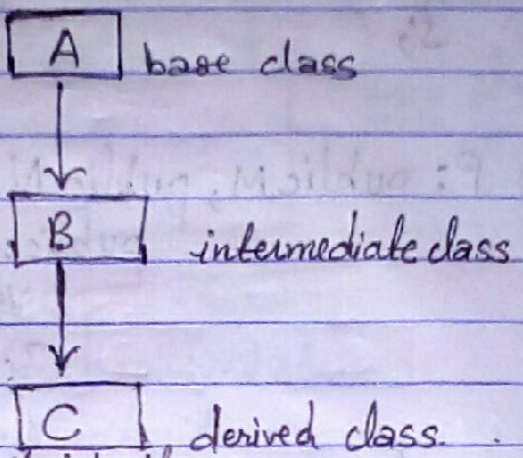
Output:

```
m=10
n=20
m*n=200
```

Date.
Page No.

iii) Multilevel inheritance:

The derived class from base class flowing through intermediate classes is called multilevel inheritance.



Multilevel inheritance

Example:-

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

class student { protected:
                int roll_number;
              public:
                void get_number(int);
                void put_number(void);
              };

void student::get_number(int a)
              {
                roll_number=a;
              }

void student::put_number()
              {
                cout<<"Roll number:"<<roll_number<<"\n";
              }
class test:public student //First -level derivation.
           { protected:
             float sub1; float sub2;
             public:
               void get_marks(float,float);
               void put_marks();
           };
```

```
void test::get_marks(float x, float y)
    {
        sub1 = x;
        sub2 = y;
    }

void test:: put_marks()
    {
        cout << "Marks in sub1 = " << sub1 << "\n";
        cout << "Marks in sub2 = " << sub2 << "\n";
    }

class result: public test //second level derivation
    {
        float total;
    public:
        void display();
    };

void result::display(){
        total = sub1+sub2;
        put_number();
        put_marks();
        cout << "Total = " << total << "\n";
    }

int main(){
    result student1;
    student1.get_number(111);
    student1.get_marks(75.0, 59.5);
    student1.display();
    return 0;
    }
```

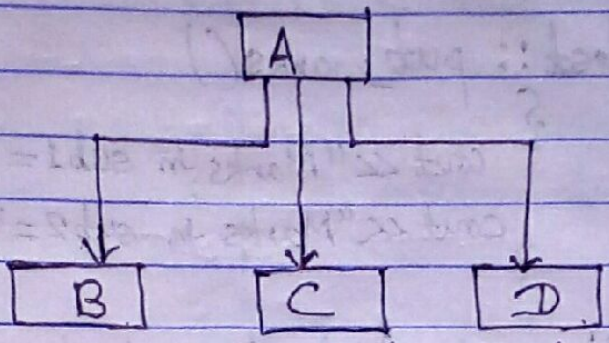
Output:

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

Date.
Page No.

iv) Hierarchiel inheritance:

If base class inherits to more than one derived classes then it is called hierarchiel inheritance.



Hierarchiel inheritance.

Example:

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

class A   //Single base class.
{ public:
    int x,y;
    void getdata(){
        cout<<"Enter values of x and y"<<"\n";
        cin>>x>>y;  }
};

class B: public A  //B is derived from base class A.
{ public:
    void product()
    { cout<<"\n Product="<<x*y;
    }
};
```

```
class C: public A //C is also derived from same base class A
{
public:
    void sum(){
    cout<<"\n Sum="<<x+y;
    }
};

int main(){
    B obj1; //object of derived class B.
    C obj2; //object of derived class C.
    obj1.getdata();
    obj1.product();
    obj2.getdata();
    obj2.sum();
    return 0;
}
```

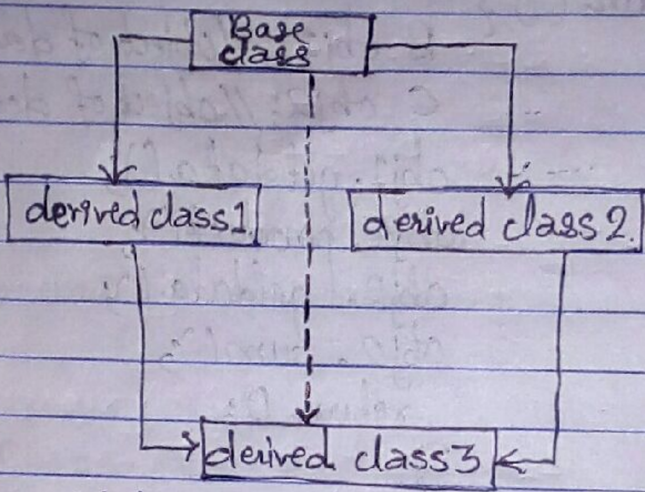
Output

```
Enter value of x and y:
2
3
Product=6
Enter value of x and y:
2
3
Sum=5
```

In this program there is only one base class A from which two class B and C are derived.

vi) Hybrid inheritance:

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.

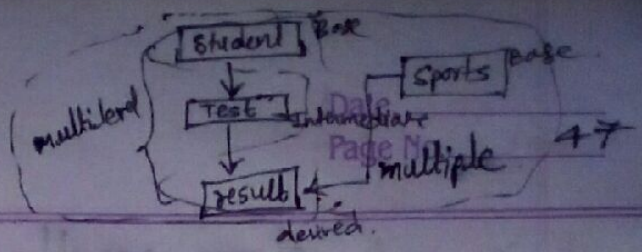


Hybrid inheritance

In this type of inheritance we need to apply two or more types of inheritance to design a program. The attributes of base class into derived class 3 in above figure will inherit twice which would create problem of duplicacy and derived class 3 would have duplicate sets of members. So, to avoid this problem we use derived class 1 and derived class 2 (intermediate classes) as virtual base class as follows:-

```
class derived_class1 : virtual public base_class
    {...
    };
class derived class 2: public virtual base_class.
    {...
    };
```

Virtual keyword scope को अगाडि पछाडि लगा सकते हैं।



47

Example:

```
#include<iostream>
using namespace std;
class student {
        protected:
        int roll_number;
public:
    void get_number(int a)
    {
        roll_number=a;
    }
    void put_number(){
cout<<"Roll No:"<<roll_number<<"\n";
    }
};

class test:public student{
        protected:
        float part1,part2;
    public:
    void get_marks(float x,float y)
    {
        part1=x; part2=y;
    }
    void put_marks(){
cout<<"Marks obtained:"<<"\n";
cout<<"Part1="<<part1<<"\n";
cout<<"Part2="<<part2<<"\n";
    }
};

class sports{
        protected:
        float score;
    public:
    void get_score(float s)
    {
        score=s;
    }
    void put_score(void){
        cout<<"sports wt:"<<score<<"\n\n";
    }
};
```

Date.
Page No.

```
class result: public test, public sports
    { float total;
      public:
          void display();
    };
void result::display(void)
    {
      total = part1+part2+score;
      put_number();
      put_marks();
      put_score();
      cout << "Total Score:" << total << "\n";
    };

int main(){
      result student_1;
      student_1.get_number(1234);
      student_1.get_marks(27.5, 33.0);
      student_1.get_score(6.0);
      student_1.display();
        return 0;
      }
```

Output

Roll No: 1234
Marks obtained:
Part 1 = 27.5
Part 2 = 33
Sports wt: 6

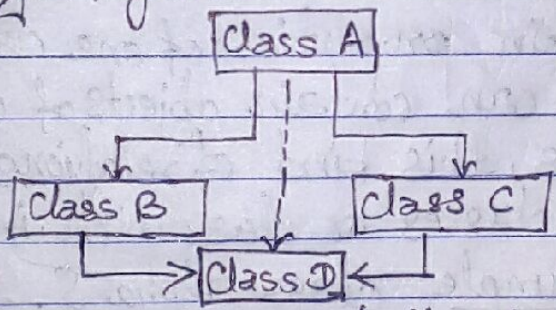
Total Score: 66.5

Abstract classes:

An abstract class is one that ~~is used to create~~ not used to create objects. An abstract class is designed only to act as a base class (to be inherited by other classes). It is a design concept in a program development and provides a base upon which other classes may be built. The classes student and sports in the previous example are abstract classes since it is not used to create any objects.

~~And~~ Ambiguity in multiple inheritance:

Ambiguity in C++ occur when a derived class have two base classes and these two base classes have one common base class. Consider the following figure.



In this figure, both class B and class C inherit class A, they both have single copy of class A. However class D inherit both class B and class C, therefore class D have two copies of class A, one from class B and another from class C.

If we need to access the data member a of class A through the object of class D, we must specify the path from which data member (i.e. a) will be accessed, whether it is from class B or class C, because compiler can't differentiate between two copies of class A in class D.

Date.
Page No.

There are two ways to avoid C++ ambiguity. One is using virtual base class as we described already in hybrid inheritance and another using scope resolution operator. Using scope resolution operator we can manually specify the path from which data member (beta) will be accessed as shown in the statements below:-

```
obj. class B::a = 10;
obj. class C::a = 100;
```

Note:- Still there are two copies of class A in class D.

Aggregation (class with in class) → Nesting of classes

Inheritance is the mechanism of deriving certain properties of one class into another. A class can contain objects of other classes as its members, this kind of relationship is called nesting of classes or class with in class.

Example

```
class alpha {...};
class beta {...};
class gama
{
    alpha a;
    beta b;
    ...
};
```

Here all objects of gamma class will contain the objects a and b. Nested object is created in two stages. First the member objects are created using their respective constructors and the other 'ordinary' members are created.

Constructors in derived class:

When a class is declared, a constructor is also declared inside the class in order to initialize data members. It is not possible to use a single constructor for more classes. Every class has its own constructor and destructor with a similar name as the class. When a class is derived from another class, it is possible to define a constructor in the derived class, and the data members of both base and derive classes can be initialized. It is not essential to declare a constructor in a base class. Thus the constructor of the derived class works for its base class; such constructors are called constructors in the derived ~~or~~ class or common constructors.

Write a simple program to initialize member variables of both base and derived class using a constructor in derived class.

This is simple program if asked in 5 marks

```
#Include<iostream.h>
#include<conio.h>
class A { protected:
            int x;
            int y;
        };
class B : private A
        { public:
            int z;
            B() { x=1, y=2, z=3;
                cout<<"x="<<x<<"y="<<y<<"z="<<z;
                }
        };
```

```
int main(){
    clrscr();
    B b;
    return 0;
}
```

Output

x=1 y=2 z=3

Explanation:- In the above program, the classes A B are declared. The class B is derived from class A. The constructor of the class B initializes member variables of both classes. Hence it acts as common constructor of both base and derived classes.

somewhat difficult if asked 10 marks in exam

Example:- // Program to show how constructors are invoked on derived class.

```
#include<iostream.h>
class alpha{
    private:
        int x;
    public:
        alpha(int i){
            x=i;
            cout<<"alpha initialized\n";
        }
        void show_x(){
            cout<<"x="<<x<<"\n";}
};
class beta{ float y;
    public:
        beta(float j){
            y=j;
            cout<<"beta initialized\n";
        }
```

constructor in base class

constructor in base class

```
void show_y(){
    cout<<"y="<<y<<"\n";
  }
};

class gamma: public beta, public alpha
{
  int m,n;
  public:
    gamma(int a, float b, int c, int d):
        alpha(a), beta(b) //function calls alpha(a) & beta(b)
    {                     // :: parameter should not include types
      m=c;
      n=d;
    cout<<"gamma initialized \n";
    }

  void show_mn(){
      cout<<"m="<<m<<"\n";
      cout<<"n="<<n<<"\n";
    }
};

int main(){
    gamma g(5, 10.75, 20, 30);   // g is object
    cout<<"\n";
     g.show_x();
     g.show_y();
     g.show_mn();

     return 0;
}
```

Note: beta is initialized first, although it appears second in derived constructor. This is becoz it has been declared first in derived class header file.

Output of Program

```
beta initialized
alpha initialized
gamma initialized

x=5
y=10.75
m=20
n=30
```

Date.
Page No.

⊛ Destructors in derived classes:

The execution of destructors is in opposite order as compared with constructors, that is, from the derived class to the base class. Thus, the sequence of the execution of destruction is that the object created last is destroyed first.

Example:

Write a program to create derived class from base classes. Use constructor and destructors.

```
#include<iostream.h>
using namespace std;

class inta{ protected:
              int i;
            public:
              int inta(){
              cout<<"Constructor inta()";
              i=1;
                 }
              ~inta() {
              cout <<"\n Destructor inta()";
                 }
           };

class flota { protected:
                float f;
              public:
                flota(){
              cout<<"\n Constructor floata()";
              f=1.5;          }
              ~flota() {
                 cout<<"\n Destructor flota()";
              };
```

Date.
Page No.

```
class chars: public inta, floata
    {
      protected:
        char c;
      public:
          void show (){
            cout<<"\n \ti="<<i;
          cout<<"\n \tf="<<f;
          cout <<"\n \tc="<<c;
                    }
        chars(){
            cout<<"\n Constructor chars()";
            c='A';
              }
            ~chars(){
          cout<<"\n Destructor chars()";
                    }
        };

int main(){
            clrscr();
            chars a;
            a.show();
            return 0;
          }
```

Output:

Constructor inta()
Constructor ~~floata()~~ flota()
Constructor chars()
i=1
f=1.5
C=A
Destructor chars()
Destructor flota()
Destructor inta()