Object Oriented Programming Using Cpp

Topic:

Inheritance

Chapter 8 Inheritance

8.1 Introduction

Inheritance is one of the most important and useful characteristics of object oriented programming language. Literally, Inheritance means adopting features by newly created thing from the existing one.

Formally, Inheritance can be defined as the process of creating a new class from one or more existing classes. The existing class is known as base class or parent class or super class whereas the newly created class is known as derived class or child class or sub class. Inheritance provides a significant advantage in terms of code reusability. Consider the diagram shown below:

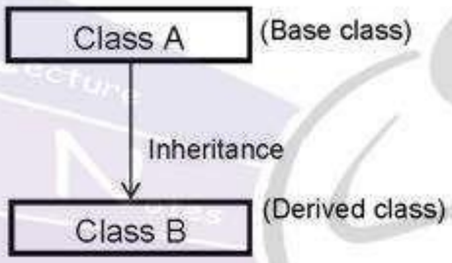

Fig: 8.1 Inheritance

In the above fig 8.1, a new class B is derived from an existing class A using the feature inheritance. Due to this the derived class B inherits the features of base class A. However, the derived class can also have its own features. Now the terms reusability means that the derived class object can access the base class members also. However, the reverse of this is not true i.e. the base class object can't access the derived class members as the base class is not aware of derivation of derived class. This chapter deals with implementation of the mechanism of inheritance along with its advantage and disadvantage.

8.2 Access Specifiers and Simple Inheritance

Till now we have discussed public and private access specifiersduring our discussion on class and object. We have seen that the public members of a class can be accessed by the object of the same class with the help of dot operator directly i.e. member function is not required when class contains only public data members. In case of private data members, class must have a public member function so that the member function may be called using object and dot operator. This public member function can access the private data members of the class. Now, here we will be discussing the third access specifier, known as protected. However, we need to discuss the process of inheritance before that to understand why protected specifier is required.

8.2.1 Simple Inheritance

In this process, a new class is created using an existing class. The newly created class is known as derived class and the existing class is known as base class. The syntax is as below:

```
class  <derived class name> : access specifer<base class name>
{
  Access specifier:
       Data members
  Access specifier:
       Member functions
};
```

In the above syntax, the derived class is the newly created class. The base class is used for the inheritance. The access specifier given before base class name tells about the method of inheritance used to derive the derived class. This specifier may be public or private. In general, the method of inheritance is public. If no specifer is mentioned then the default specifier will be private. So, the user must take care of writing public as access specifier to indicate the system that the method of inheritance is public inheritance.

Here, the derived class inherits the features of base class due to inheritance. Additionally, derived class also has its own data members as well as member functions.

8.2.2 Methods of Inheritance

A new class can be derived from an existing class in one of the following two methods:

1. Public Inheritance

2. Private Inheritance

1. Public Inheritance: When a class is derived publicly from an existing class, then all the public members of base class behaves as public members of derived class. So, they can be accessed directly by the derived class object with the help of dot operator. However, if the base class contains private members and the method of inheritance is public inheritance then the derived class can access the private members of base class only with the help of member function of base class. The public derivation does not allow the derived class to access the private members of the base class directly. So, in this case base class must have a public member function which can be accessed by the derived class object. That member function can access the private data members of base class.

Program 44: Write a C++ program to illustrate the concept of simple inheritance using public inheritance method.

Code:

```
#include<iostream.h>
#include<conio.h>
class  ONE                    //Base Class
{
        public:
                int     x;
};
class  TWO : public   ONE       // Derived class
{
        public:
                int     y;
};
void main()
{
        TWO  ob;
        cout<<"Enter the value of x and y ";
        cin>>ob.x;    // Access to base class data member
        cin>>ob.y;    // Access to derived class data member
```

```
            cout<<"x="<<ob.x<<endl<<"y="<<ob.y;
        }
```

O/P:

```
        Enter the value of x and y
        10      20
        x=10
        y=20
```

In the above example, class TWO is derived from class ONE using public inheritance. So, the public member of base class i.e. x behaves as public member of derived class. In main(), using derived class object, ob, we have provided the input for x and displayed them. In this example, we have not taken any member function as derived class also does not have any private member so as the base class.

Program 45: Write a C++ program to illustrate the concept of simple inheritance using public inheritance method. The base class data member must be private.

Code:

```
        #include<iostream.h>
        #include<conio.h>
        class  ONE                          //Base Class
        {
                    int     x;
            public:
                    void  fun()
                    {
                            cout<<"Enter the value of x";
                            cin>>x;
                            cout<<"x="<<x<<endl;
                    }
        };
```

```
class  TWO : public    ONE        // Derived class
{
        int      y;
        public:
                void data()
                {
                        cout<<"Enter the value of y";
                        cin>>y;
                        cout<<"y="<<y<<endl;
                }
};
void main()
{
        TWO  ob;
        ob.data();     //Access to derived class member function
        ob.fun();      //Access to base class public member function
}
```

O/P:

```
Enter the value of y
20
y=20
Enter the value of x
10
x=10
```

In the above example, the base class contains a private data member, x. Now to access this private data member we must have to take a public member function of base class as public inheritance does not allow derived class object to access the private data member of base class. In main(), the derived class object is used to call the base class public member function, fun(). This fun() function can access the private data member, x of base class. However, it is strongly advised to not use any member function inside the base class because in inheritance the base class acts as a skeleton based on which other classes may be designed. So, they prohibit use of member function in base class. Now, this will arise a genuine query that without base class member function, how the private data members of base class can be accessed. This query has been addressed latter in this chapter.

2. Private Inheritance: This is another method using which a new class can be derived from an existing class. This method is very rarely used as it creates lot of complexities for the programmer.

When a class is derived privately, the derived class object has no permission to access even the public members of base class. In this case, the public members of the base class can only be accessed using the public member function of the derived class. Though private inheritance provides desired security, it is not frequently used due to the complexity in accessing the base class members.

Program 46: Write a C++ program to illustrate the concept of simple inheritance using private inheritance method. The base class data member must be private.

Code:

```
#include<iostream.h>
#include<conio.h>
class  ONE                    //Base Class
{
            int     x;
      public:
            void fun()
            {
                  cout<<"Enter the value of x";
                  cin>>x;
                  cout<<"x="<<x<<endl;
```

```
                }
        };
        class  TWO : private   ONE        // Derived class
        {
                int      y;
                public:
                        void data()
                        {
                           fun(); //Accessing of base class public member function
                        cout<<"Enter the value of y";
                        cin>>y;
                        cout<<"y="<<y<<endl;
                        }
        };
        void main()
        {
                TWO  ob;
                ob.data();     //Access to derived class member function
        }
```

O/P:

```
Enter the value of x
10
x=10
Enter the value of y
20
y=20
```

In the above example, class TWO is derived from class ONE using private inheritance method. Due to this reason, the derived class object is not able to access even the public member of base class. We can see that in main() the derived class object, ob calls the data() function, public member function of derived class. This data() function calls the public member function of base class which can access the private data member of base class. So, we can conclude that in private inheritance, even the public members of base class can be accessed by the public member function of derived class. Due to this complexity, private inheritance method is not frequently used.

8.2.3 Protected data with private inheritance

In the above case, we saw that the derived class must have a public member function which can access the public member function of base class. This public member function can access the private member of base class. This increase the complexity of program as the length of the code increases.

To overcome the above mentioned problem, C++ provides a new access specifier known as protected. Protected specifier is same as private but it allows the derived class to access the private members directly. Actually protected data of a class is public for derived class only but they are private for other classes.

Program 47: Write a program in C++ to explain the use of protected data in private inheritance.

Code:

```
#include<iostream.h>

#include<conio.h>

class  BASE

{

        protected:

                int  num1;

};

Class  DERIVED  :  private  BASE

{

        int  num2;

        public:
                void fun()
                {
                        num1=10;
                        num2=20;
                        cout<<"num1="<<num1<<endl<<"num2="<<num2;
                }
        };
        void main()
        {
                DERIVED  obj;
                obj.fun();
        }
```

O/P:

```
        num1=10
        num2=20
```

Note: The most general practice is to define base class with only protected data members. The method of inheritance is mostly public.

8.3 Types of Inheritance

The process of inheritance is broadly classified into following six categories:

i. Single Inheritance

ii. Multilevel Inheritance

iii. Multiple Inheritance

iv. Hybrid Inheritance

v. Hierarchical Inheritance

vi. Multipath Inheritance

i. Single Inheritance: The process to derive a new class from an existing class is known as single inheritance. The new class is known as base class and the existing class is known as derived class.

Consider the diagram shown below:

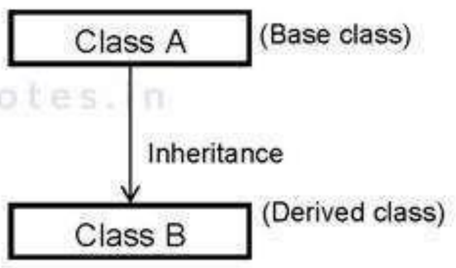

Fig 8.2 : Single Inheritance

Here, class B is derived from class A. Due to this inheritance, class B adopts the features of base class A. Additionally, class B also contains its own features. The process of inheritance enables the derived class B to access the base class members.

Program 48: Write a program in C++ to add two integers single inheritance.

Code:

```
#include<iostream.h>
#include<conio.h>
class  ONE
{
        protected:
                int     n1;
};
Class  TWO : public   ONE          // Single Inheritance
{
        int     n2;
        public:
                void  input()
                {
                        cout<<"Enter n1 and n2";
                        cin>>n1>>n2;
                }
                void  sum()
                {
                        int   sum=0;
                        sum=n1 + n2;
                        cout<<"Sum is "<<sum;
                }
        };
        void  main()
        {
                TWO  t1;
                t1.input();
                t1.sum();
        }
```

O/P:

```
Enter n1 and n2
10      20
Sum is 30
```

ii. Multilevel Inheritance: When the process of single inheritance is extended up to minimum 2 levels, the concept is known as multilevel inheritance. The process of multilevel inheritance involves a new terminology known as "Intermediate class." This is discussed in detail with the following example.

Consider the diagram shown below:

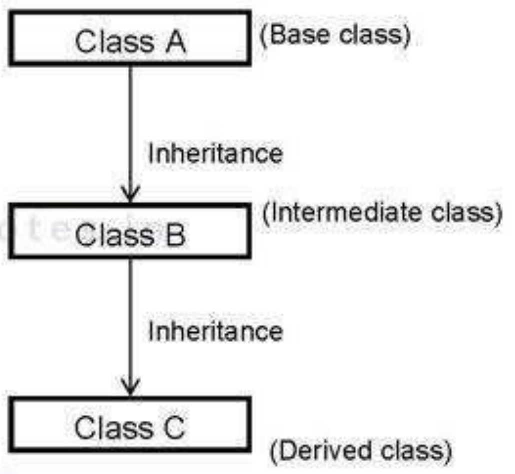

Fig 8.3 : Multilevel Inheritance

Here, class B is derived from class A. Due to this inheritance, class B adopts the features of base class A. Additionally, class B also contains its own features. Further, a new class, class C is derived from class B. Due to this, class C adopts the features of class B as well as features of class A (As B contains A features). Additionally, class C may have its own features. Here, we can see that class B acts as base derived class for class A and it also acts as base class for class C i.e. it performs dual responsibility due to which it is known as intermediate class.

Program 48: Write a program in C++ to add two integers multilevel inheritance.

Code:

```
#include<iostream.h>
#include<conio.h>
class  ONE
{
        protected:
                int     n1;
};
Class  TWO : public   ONE
{
```

```
          protected:
          int      n2;
      };
      class  THREE : public  TWO
      {
          public:
                    void  input()
                    {
                              cout<<"Enter n1 and n2";
                              cin>>n1>>n2;
                    }
                    void  sum()
                    {
                              int   sum=0;
                              sum=n1 + n2;
                              cout<<"Sum is "<<sum;
                    }
      };
      void  main()
      {
          THREE       t1;
          t1.input();
          t1.sum();
      }
```

O/P:

```
Enter n1 and n2
10      20
```

```
Sum is 30
```

iii. Multiple Inheritance: When a new class is derived from more than one base class, the process is known as multiple inheritance. Here, the minimum number of base class is 2. If the number of base class goes below two then it becomes the case of single inheritance. Hence, to implement multiple inheritance user must have to take minimum 2 base classes from where the new class is derived.

Consider the diagram shown below:

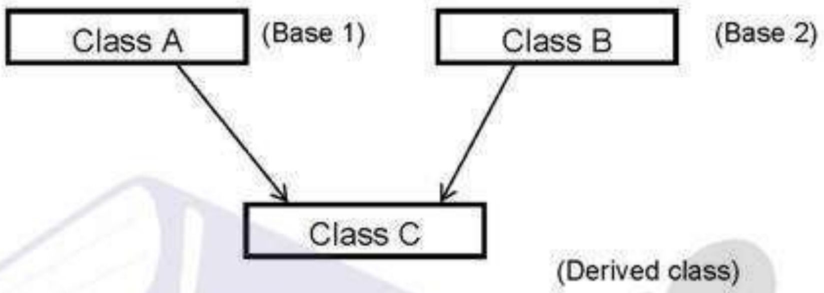

Fig 8.4 : Multiple Inheritance

Here, class C is derived from two base classes i.e. class A and class B. Due to this, class C adopts the features of base classes A and B. Additionally, class C may have its own features. In multiple inheritance, one important thing to take care is to mention the method of inheritance for each base class involved in the process

Program 48: Write a program in C++ to add two integers multilple inheritance.

Code:

```
#include<iostream.h>

#include<conio.h>

class  ONE

{

        protected:

                int      n1;

};

Class  TWO

{

        protected:
            int     n2;
    };
    class  THREE : public  ONE, public  TWO
    {
            public:
                    void  input()
                    {
                            cout<<"Enter n1 and n2";
                            cin>>n1>>n2;
                    }
                    void  sum()
                    {
                            int   sum=0;
                            sum=n1 + n2;
                            cout<<"Sum is "<<sum;
                    }
    };
    void  main()
    {
            THREE       t1;
            t1.input();
            t1.sum();
    }
```

O/P:

```
    Enter n1 and n2
    10     20
    Sum is 30
```

Note: In the above program, it is compulsory to mention the method of inheritance as public for both base classes. Failing to do so will make the process as private inheritance for which the programming will be a bit different.

iv. Hybrid Inheritance: The proper combination of one or more type of inheritance happening together is known as hybrid inheritance. The following diagram explains the concept in a better way.

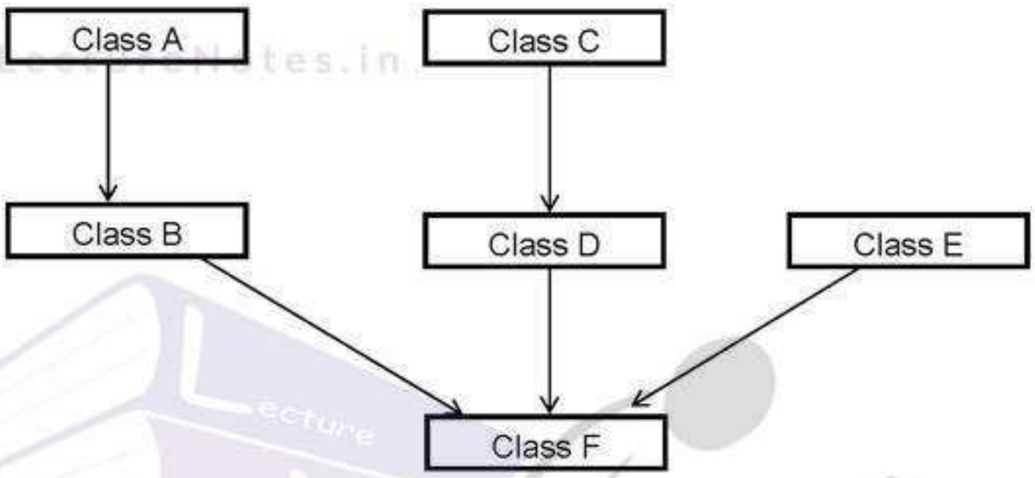

Fig 8.5: Hybrid Inheritance

In the above diagram, the derivation of class B from class A is the single inheritance. The derivation of class F from class D which is derived from class C is the multilevel inheritance. Now, the derivation of class F from class B, D and E is the multiple inheritance.

Program 49: Write a program in C++ to explain the process of hybrid inheritance.

Code:

```
#include<iostream.h>

#include<conio.h>

class  A

{

        protected:

                int     n1;

};
```

```
class  B : public  A
{
        protected:
                int     n2;
};
class  C
{
        protected:
                int     n3;
};
class  D : public  A
{
        protected:
                int     n4;
};
class  E
{
        protected:
                int     n5;
};
class  F : public     B, public     D, public     E
{
        public:
                void   fun()
                {
                        cout<<"Enter five integers";
                        cin>>n1>>n2>>n3>>n4>>n5;
```

```
                    int        sum=0;
                    sum=n1+n2+n3+n4+n5;
                    cout<<"The entered numbers are "<<endl;
                    cout<<"n1="<<n1<<endl;
                    cout<<"n2="<<n2<<endl;
                    cout<<"n3="<<n3<<endl;
                    cout<<"n4="<<n4<<endl;
                    cout<<"n5="<<n5<<endl;
                    cout<<"Sum="<<sum<<endl;
          }
     };
     void  main()
     {
          F      obj;
          obj.fun();
     }
```

O/P:

```
Enter five integers
10    20    30    40       50
n1=10
n2=20
n3=30
n4=40
n5=50
Sum=150
```

v. Hierarchical Inheritance: In this type of inheritance more than one derived classes are present. This inheritance is little bit complicated as it involves lots of base classes and derived classes. Due to this reason, this inheritance is not frequently used. Consider the diagram shown below:

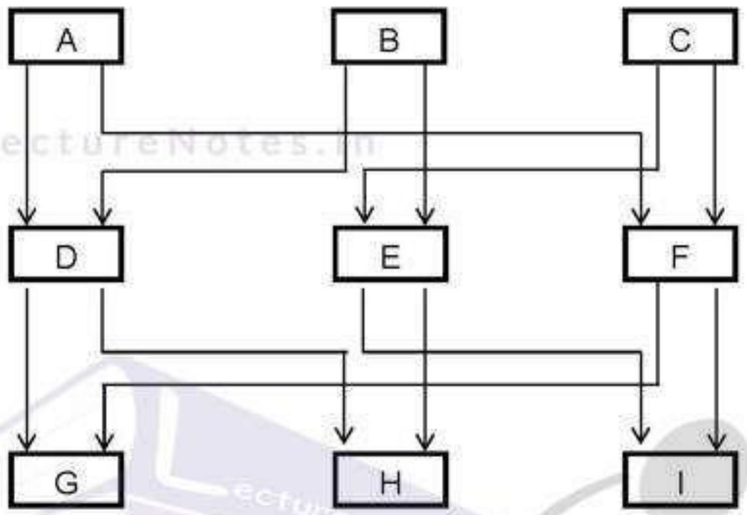

Fig 8.6: Hierarchical Inheritance

In the above diagram, each box represents a class and arrow represents inheritance (top to bottom). From the above diagram, an important note is to be made which will represent inheritance between two classes. The information is mentioned in table 8.1 as shown below:

Base class	Derived class
A	D, F
B	D, E
C	E, F
D	G, H
E	H, I
F	G, I

Table 8.1: Base and derived classes

From table 8.1, we can conclude that in fig 8.6 classes A, B and C behave base classes. Classes D, E and F behave as intermediate classes and classes G, H and I behave as derived classes.

Now, class D and F has been derived from base class A. Similarly we can see that class D has been derived from two base classes A and B. This shows that hierarchical inheritance involves more than one base and derived classes.

Program 50: Write a program in C++ to illustrate the concept of hierarchical inheritance

Code:

```
#include<iostream.h>
#include<conio.h>
class A
{
        public:
        A()
        {
                cout<<"Class A constructor"<<endl;
        }
};
class B
{
        public:
        B()
        {
                cout<<"Class B constructor"<<endl;
        }
};
class C
{
        public:
        C()
        {
                cout<<"Class C constructor"<<endl;
        }
};
class D:public A, public B
{
        public:
        D()
        {
                cout<<"Class D constructor"<<endl;
        }
};
class E: public B, public C
{
        public:
        E()
        {
                cout<<"Class E constructor"<<endl;
        }
};
class F:public A, public C
{
        public:
        F()
        {
                cout<<"Class F constructor"<<endl;
        }
};
class G:public D, public F
```

```
{
        public:
        G()
        {
                cout<<"Class G constructor"<<endl;
        }
};
class H:public D, public E
{
        public:
        H()
        {
                cout<<"Class H constructor"<<endl;
        }
};
class I:public E,public F
{
        public:
        I()
        {
                cout<<"Class I constructor"<<endl;
        }
};
void main()
{
        G ob1;
        H ob2;
        I ob3;
        getch();
    }
```

O/P:

```
Class A constructor
Class B constructor
Class D constructor
Class A constructor
Class C constructor
Class F constructor
Class G constructor
Class A constructor
Class B constructor
Class D constructor
Class B constructor
Class C constructor
Class E constructor
Class H constructor
Class B constructor
Class C constructor
Class E constructor
Class A constructor
Class C constructor
Class F constructor
Class I constructor
```

Note: In program 50, each class contains a constructor. Now one important thing here to understand is the execution of constructor in inheritance.

First of all, constructors and destructors are not inherited. However, they are executed. In this process the order of constructor and destructor execution is as below:

i. First base class constructor is executed

ii. After that derived class constructor is executed.

iii. Destructor executes in reverse order i.e. first derived class destructor then base class destructor

vi. Multipath Inheritance: In this type of inheritance a class is derived from more than two or more classes, which are derived from the same base class i.e. The final derived class is derived from some intermediate classes which are derived from same base class. It is also known as diamond inheritance.

Following diagram explains the concept of multipath inheritance.

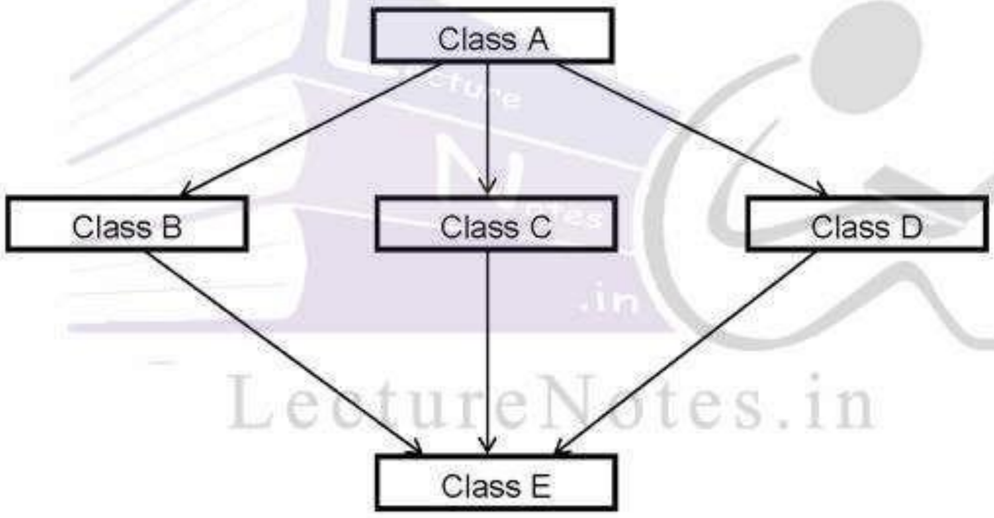

Fig 8.7: Multipath Inheritance

In the above diagram, we can see that class B, C and D has been derived from same base class A. Further class E has been derived from class B, C and D. So, here class A acts as common base class for classes B, C and D which act as intermediate classes. Class E is the final derived class.