Chapter 8

Inheritance

Introduction

Reusability is another important feature of OOP's. It is always better if we reuse something that already exist rather than trying to create the same thing again. It could not only saves the time also increase the reability.

C++ strongly supports the concept of reusability. The C++ classes can be reuse in several ways. Once a class has be written and tested, it can be adopted by other classes as per their requirement. This is done by creating new classes and reusing the properties of existing classes.

The mechanism of deriving a new class from an old one (existing) is called Inheritance. The old class from which properties are inherited is known as Base class and the new class which inherit the properties is known as Derived class.

Defining Derived Classes

A derived class is defined be specifying its relationship with the base class in addition to its own details. The general form of defining a derived class is as follows

```
class Derived Class : Visibility  Base Class
      Name              Mode      Name
{
         Member of
         Derived Class
};
```

The ':' 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 specify weather the features of the base class are privately derived or publicly derived.

When a base class is privately inherited by a derived class, public member of a base class become private members of the derived class. Therefore the public members of the class can only be access of the member function of the derived class.

On the other hand when the base class is publicly inherited, public member of the base class become public member of the derived class and therefore they are accessible to the objects of the derived class.

e.g.

```
class personal
{
   private :
        int rno;
        char name[30];
```

```
    public :
        void getdata( )
        {
            cout<<"Enter the roll no & Per = ";
            cin>>rno>>name;
        }
        void putdata( )
        {
            cout<<"Roll no = "<<rno;
            cout<<"Name = "<<name;
        }
};

class mark : public personal
{
        int m1,m2,m3,total;
        float per;
    public :
        void getmark( )
        {
            cout<<"Enter 3 Sub Marks = ";
            cin>>m1>>m2>>m3;
        }
        void putmark( )
        {
            cout<<"Mark1 = "<<m1;
            cout<<"Mark2 = "<<m2;
            cout<<"Mark3 = "<<m3;
            cout<<"Total = "<<total;
            cout<<"Per = "<<per;
        }
};
```

Note

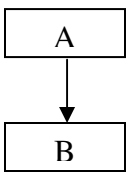
In both cases the private members are not inherited. Therefore the private member of the base class will never become the members of its derived class.

Types of Inheritance

There are various types of inheritance is as follows

1) Single Inheritance

When a derived class has only one base class, such inheritance is known as Single Inheritance.

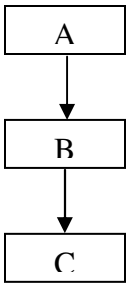


Base Class

Derived Class

2) Multilevel Inheritance

The mechanism of deriving a class from another derived class is known as Multilevel Inheritance.



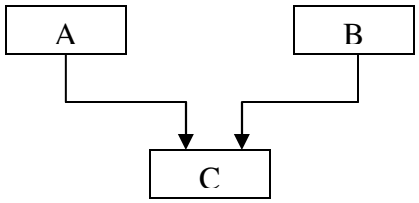
Pure Base Class

Intermediate Base Class

Derived Class

3) Multiple Inheritance

When a derived class has several vase classes is known as Multiple Inheritance.

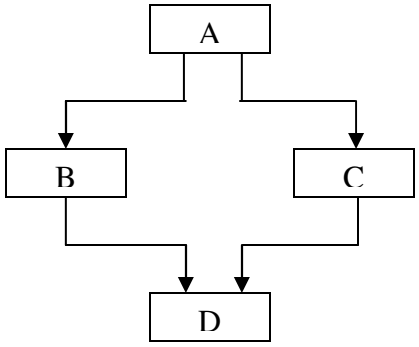


Base Classes

Derived Class

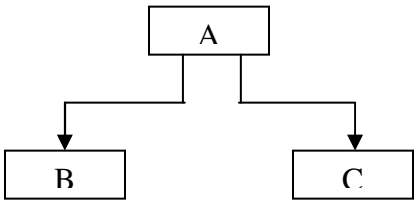
4) Hybrid Inheritance

When in the derivation of a class so many types of inheritance process are involved, such inheritance is called as Hybrid Inheritance.



5) Hierarchical Inheritance

When the properties of one class are inherited by more than one classes i.e. from a single base class many classes are derived, such inheritance is known as Hierarchical Inheritance.



Base Class

Derived Classes

Inheritance of Private Members

In any type of inheritance the private members of the base class can not be inherited. But if we want to inherit the private member also then we have to make private member also then we have to make private members as public. The facility of the private members of the class will not be useful if the data members become public i.e. the data hiding property provided by C++ will not be useful.

C++ provide third visibility mode known as Protected. The data members or member function declared as protected will not be access by the object of the class but the members are inherited some other class. In public derivation of a class the protected member protected and becomes private in the private derivation of a class.

Base Class Visibility	Derived Class Visibility	
	Public Derivation	**Private Derivation**
1) Private	Not Inherited	Not Inherited
2) Protected	Protected	Private
3) Public	Public	Private

Write a Program to Demonstration of Single Inheritance in Public Derivation.

```
#include<iostream.h>
#include<conio.h>

class personal
{
      int rno;
      char name[30];
  public:
      void getdata()
      {
       cout<<"Enter Roll no & Name = ";
       cin>>rno>>name;
      }
      void putdata()
      {
       cout<<"Roll no = "<<rno<<endl;
       cout<<"Name = "<<name<<endl;
      }
};

class mark : public personal
{
      int m1,m2,m3,total;
      float per;
  public:
      void getmark()
      {
       cout<<"Enter 3 sub Marks = ";
       cin>>m1>>m2>>m3;
      }
```

```
      void putmark()
      {
       cout<<"Mark1 = "<<m1<<endl
           <<"Mark2 = "<<m2<<endl
           <<"Mark3 = "<<m3<<endl
           <<"Total = "<<total<<endl
           <<"Per = "<<per;
      }
      void cal()
      {
       total=m1+m2+m2;
       per=total/3;
      }
};

main()
{
mark p;
clrscr();
p.getdata();
p.getmark();
p.putdata();
p.cal();
p.putmark();
getch();
return(0);
}
```

Write a Program to Demonstration of Single Inheritance in Private Derivation.

```
#include<iostream.h>
#include<conio.h>

class personal
{
  private:
      int rno;
      char name[30];
  public:
      void getdata()
      {
       cout<<"Enter Roll no & Name = ";
       cin>>rno>>name;
      }
      void putdata()
      {
       cout<<"Roll no = "<<rno<<endl;
       cout<<"Name = "<<name<<endl;
      }
};
```

```
class mark : private personal
{
      int m1,m2,m3,total;
      float per;
  public:
      void getmark()
      {
       getdata();
       cout<<"Enter 3 sub Marks = ";
       cin>>m1>>m2>>m3;
      }
      void putmark()
      {
       putdata();
       cout<<"Mark1 = "<<m1<<endl
           <<"Mark2 = "<<m2<<endl
           <<"Mark3 = "<<m3<<endl
           <<"Total = "<<total<<endl
           <<"Per = "<<per;
      }
      void cal()
      {
       total=m1+m2+m2;
       per=total/3;
      }
};

main()
{
mark p;
clrscr();
p.getmark();
p.cal();
p.putmark();
getch();
return(0);
}
```

Multilevel Inheritance

When a class is derived from another derived class known as Multilevel Inheritance. The derived class whose properties are inherited is known as intermediate base class.

Write a Program to Demonstration of Multilevel Inheritance.

```
#include<iostream.h>
#include<conio.h>

class rollno
{
      int rno;
```

```
  public:
      void getroll()
      {
       cout<<"Enter Roll no = ";
       cin>>rno;
      }
      void putroll()
      {
       cout<<"Roll no = "<<rno<<endl;
      }
};

class mark : public rollno
{
  protected:
      int m1,m2,m3;
  public:
      void getmark()
      {
       cout<<"Enter 3 sub Marks = ";
       cin>>m1>>m2>>m3;
      }
      void putmark()
      {
       cout<<"Mark1 = "<<m1<<endl
           <<"Mark2 = "<<m2<<endl
           <<"Mark3 = "<<m3<<endl;
      }
};

class result : public mark
{
  private:
      int total;
      float per;
  public:
      void display()
      {
       total=m1+m2+m3;
       per=total/3;
       cout<<"Total = "<<total<<endl
           <<"Per = "<<per;
      }
};

main()
{
result p;
clrscr();
p.getroll();
p.getmark();
p.putroll();
```

```
p.putmark();
p.display();
getch();
return(0);
}
```

Multiple Inheritance

When a class is derived from more than one base class such inheritance is known as Multiple Inheritance.

Syntax

```
class Derived Class  :  Visibility  Base Class 1, Visibility  Base Class 2
      Name              Mode                      Mode
{
        Member of
        Derived Class
};
```

Write a Program to Demonstration of Multiple Inheritance.

```
#include<iostream.h>
#include<conio.h>

class rollno
{
  private:
      int rno;
  public:
      void getroll()
      {
       cout<<"Enter Roll no = ";
       cin>>rno;
      }
      void putroll()
      {
       cout<<"Roll no = "<<rno<<endl;
      }
};

class mark
{
  protected:
      int m1,m2,m3;
  public:
      void getmark()
      {
       cout<<"Enter 3 sub Marks = ";
       cin>>m1>>m2>>m3;
      }
```

```
      void putmark()
      {
       cout<<"Mark1 = "<<m1<<endl
           <<"Mark2 = "<<m2<<endl
           <<"Mark3 = "<<m3<<endl;
      }
};

class result : public rollno,public mark
{
  private:
      int total;
      float per;
  public:
      void display()
      {
       total=m1+m2+m3;
       per=total/3;
       cout<<"Total = "<<total<<endl
           <<"Per = "<<per;
      }
};

main()
{
result p;
clrscr();
p.getroll();
p.getmark();
p.putroll();
p.putmark();
p.display();
getch();
return(0);
}
```

Hierarchical Inheritance

When the properties of the class are inherited by more then one derived classes in other words when a class work as a base for more than one derived classes such inheritance is known as Hierarchical Inheritance.

Write a Program to Demonstration of Hierarchical Inheritance.

```
#include<iostream.h>
#include<conio.h>

class rollno
{
  private:
      int rno;
```

```
  public:
      void getroll()
      {
       cout<<"Enter Roll no = ";
       cin>>rno;
      }
      void putroll()
      {
       cout<<"Roll no = "<<rno<<endl;
      }
};

class mark : public rollno
{
  private:
      int m1,m2,m3;
  public:
      void getmark()
      {
       cout<<"Enter 3 sub Marks = ";
       cin>>m1>>m2>>m3;
      }
      void putmark()
      {
       cout<<"Mark1 = "<<m1<<endl
           <<"Mark2 = "<<m2<<endl
           <<"Mark3 = "<<m3<<endl;
      }
};

class sport : public rollno
{
  private:
      int smark;
  public:
      void getdata()
      {
       cout<<"Enter Sport Mark = ";
       cin>>smark;
      }
      void putdata()
      {
       cout<<"Sport Mark = "<<smark<<endl;
      }
};

main()
{
mark x;
clrscr();
x.getroll();
x.getmark();
```

```
x.putroll();
x.putmark();
sport y;
y.getroll();
y.getdata();
y.putroll();
y.putdata();
getch();
return(0);
}
```

Write a Program to Demonstration of Hybrid Inheritance.

```
#include<iostream.h>
#include<conio.h>

class rollno
{
  private:
      int rno;
  public:
      void getroll()
      {
       cout<<"Enter Roll no = ";
       cin>>rno;
      }
      void putroll()
      {
       cout<<"Roll no = "<<rno<<endl;
      }
};

class mark : public rollno
{
  protected:
      int m1,m2,m3;
  public:
      void getmark()
      {
       cout<<"Enter 3 sub Marks = ";
       cin>>m1>>m2>>m3;
      }
      void putmark()
      {
       cout<<"Mark1 = "<<m1<<endl
           <<"Mark2 = "<<m2<<endl
           <<"Mark3 = "<<m3<<endl;
      }
};
```

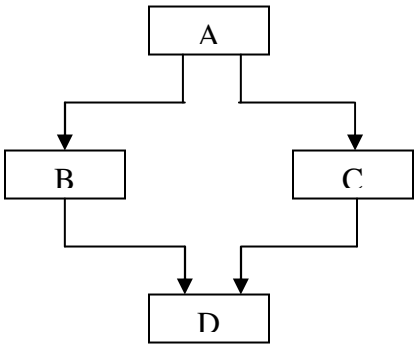
```
class sport : public rollno
{
  protected:
      int smark;
  public:
      void getdata()
      {
       cout<<"Enter Sport Mark = ";
       cin>>smark;
      }
      void putdata()
      {
       cout<<"Sport Mark = "<<smark<<endl;
      }
};

class result : public mark,public sport
{
  private:
      int total;
      float per;
  public:
      void display()
      {
       total=m1+m2+m3+smark;
       per=total/3;
       cout<<"Total = "<<total<<endl
           <<"Per = "<<per;
      }
};

main()
{
result x;
clrscr();
x.getroll();
x.getmark();
x.getdata();
x.putroll();
x.putmark();
x.putdata();
x.display();
getch();
return(0);
}
```

Virtual Base Class

In hybrid inheritance the derivation of a class involves multiple, multilevel & hierarchical inheritance. While such kind of heritance the properties of any class may get duplicate as shown in the following figure.



Assume that there is a data members X in class A. The properties of A are inherited by the classes B & C i.e. the data members X will be in the class B & C. The properties of B & C are inherited by D so the data members X is duplicated in the class D from the path B to D and from C to D. This duplication will create ambiguity in the class D.

The duplication of inherited members due to the multiple path 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
{
        Member of
        Class A
};

class B : public virtual A
{
        Member of
        Class B
};

class C : public virtual A
{
        Member of
        Class C
};

class D : public B, public C
{
        Member of
        Class D
};
```

Write a Program to Demonstration of Virtual Base Class.

```
#include<iostream.h>
#include<conio.h>

class rollno
{
  private:
      int rno;
  public:
      void getroll()
      {
       cout<<"Enter Roll no = ";
       cin>>rno;
      }
      void putroll()
      {
       cout<<"Roll no = "<<rno<<endl;
      }
};

class mark : public virtual rollno
{
  protected:
      int m1,m2,m3;
  public:
      void getmark()
      {
       cout<<"Enter 3 sub Marks = ";
       cin>>m1>>m2>>m3;
      }
      void putmark()
      {
       cout<<"Mark1 = "<<m1<<endl
           <<"Mark2 = "<<m2<<endl
           <<"Mark3 = "<<m3<<endl;
      }
};

class sport : virtual public rollno
{
  protected:
      int smark;
  public:
      void getdata()
      {
       cout<<"Enter Sport Mark = ";
       cin>>smark;
      }
```

```
      void putdata()
      {
       cout<<"Sport Mark = "<<smark<<endl;
      }
};

class result : public mark,public sport
{
  private:
      int total;
      float per;
  public:
      void display()
      {
       total=m1+m2+m3+smark;
       per=total/4;
       cout<<"Total = "<<total<<endl
           <<"Per = "<<per;
      }
};

main()
{
result x;
clrscr();
x.getroll();
x.getmark();
x.getdata();
x.putroll();
x.putmark();
x.putdata();
x.display();
getch();
return(0);
}
```

Abstract Classes

An abstract class is one that is 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 program development and provides a base upon which other classes may be built.

Containership: Classes within Classes (Nesting of Classes)

Inheritance is the mechanism of deriving certain properties of one class into another. We have seen in detail how this is implemented using the concept of derived classes. C++ supports yet another way of inheriting properties of one class into another. This approach takes a view an object can be a collection of many other objects. That is, a class can contain objects of other classes as its member.

e.g.

```
class alpha
{
  ……
}

class beta
{
  ……
}

class gamma
{
   alpha a;
   beta b;
   …….
};
```

All objects of gamma class will contain the object a and b. This kind of relationship is called Containership or Nesting.

Creation of an object that contain another object is very different than the creation of an independent object. An independent object is created by its constructor when it is declared with arguments. On the other hand , a nested object is created in two stages. First, the member objects are created using their respective constructors and then the other 'ordinary' members are created. This means, constructors of all the member objects should be called before its own constructor body is executed. This is accomplished using an initialization list in the constructor of the nested class.

e.g

```
class gamma
{
      …..
      alpha a;
      beta b;
   public :
      gamma (arglist) : a(arglist1),b(arglist2)
      {
         // constructor body
      }
};

gamma(int x,int y,float z) : a(x),b(x,z)
{
         Assignment section
}
```

We can use as many member objects as are required in a class. For each member object we add a constructor call in the initializer list. The constructor of the member objects are called in the order in which they are declared in the nested class.

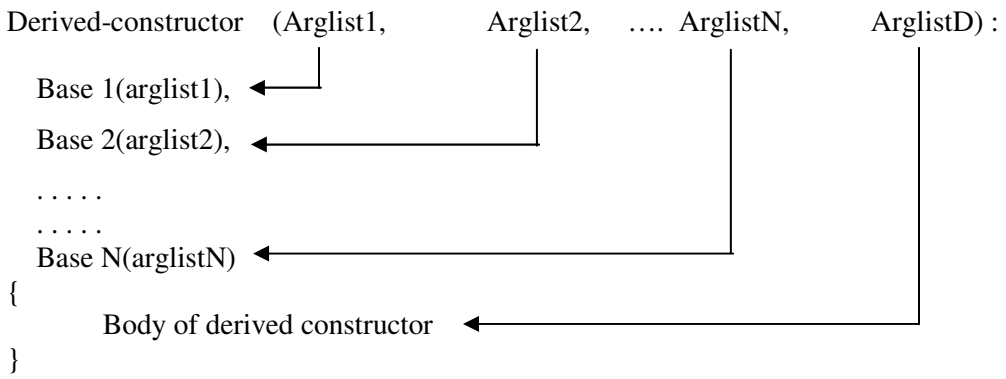
Constructors in Derived Classes

However, if any base class contains a constructor with one or more arguments, then it is mandatory for the derived class to have a constructor and pass the arguments to the base class constructor. Remember, while applying inheritance we usually create objects using the derived class. Thus, it makes sense for the derived class to pass arguments to the base class constructor. When both the derived and base classes contain constructors, the base constructor is executed first and then the constructor in the derived class is executed.

In case of multiple inheritance, the base classes are constructed in the order in which they appear in the declaration of the derived class. Similarly, in a multilevel inheritance, the constructors will be executed in the order of inheritance. Since the derived class takes the responsibility of supplying initial values to its base classes, we supply the initial values that are required by all the classes together when a derived class object is declared.

The constructor of the derived class receives the entire list of values as its arguments and passes them on to the base constructors in the order in which they are declared in the derived class. The base constructor are called and executed before executing the statements in the body of the derived constructor.

Syntax



The header line of derived-constructor function contains two parts separated by a colon (:). The first part provides the declaration of the arguments that are passed to the derived-constructor and the second part lists the function calls to the base constructors.

Base1(arglist1), Base2(arglist2) . . . are function calls to base constructors base1( ), base2( ), ... and therefore arglist1, arglist2, …. etc. represent the actual parameters that are passed to the base constructors. Arglist1 through ArglistN are the argument declarations for the base constructors base1 through baseN. ArglistD provides the parameters that are necessary to initialize the members of the derived class.

e.g.

```
D(int a1,int a2,float b1,float b2,int d1) :
    A(a1,a2),    // Call to constructor A
    B(b1,b2)     // Call to constructor B
{
   d = d1;       // executes its own body
}
```

The above constructor function invoked as follows

D objD(5,12,2.5,7.55,30);

The constructors for virtual base classes are invoked before any non-virtual base classes. If there are multiple virtual base classes, they are invoked in the order in which they are declared. Any non-virtual bases are then constructed before the derived class constructor is executed.

Write a Program to Demonstration of Constructors in Derived Classes.

```
#include<iostream.h>
#include<conio.h>

class alpha
{
      int x;
  public:
      alpha(int i)
      {
       x=i;
       cout<<"Alpha Initialized"<<endl;
      }
      void showx()
      {
       cout<<"X = "<<x<<endl;
      }
};

class beta
{
      float y;
  public:
      beta(float j)
      {
       y=j;
       cout<<"Beta Initialized"<<endl;
      }
      void showy()
      {
       cout<<"Y = "<<y<<endl;
      }
};

class gamma : public beta,public alpha
{
      int m,n;
  public:
      gamma(int a,float b,int c,int d):
     alpha(a),beta(b)
      {
       m=c;
```

```
        n=d;
        cout<<"Gamma Initialized"<<endl;
        }
        void showmn()
        {
        cout<<"M = "<<m<<endl
            <<"N = "<<n;
        }
};

main()
{
clrscr();
gamma g(5,10.57,20,40);
g.showx();
g.showy();
g.showmn();
getch();
return(0);
}
```

Class Hierarchies

Class hierarchy consists of a base class and derived classes. When a derived class has a single base class, it is known as single inheritance. When a derived class has more than one base class (multiple inheritance), it is known as class network.

A collection of classes, some of which are derived from others. A class network is a class hierarchy generalized to allow for multiple inheritance. It is sometimes known as forest model of classes.