UNIT 11 : INHERITANCE

UNIT STRUCTURE

11.1 LEARNING OBJECTIVES

After going through this unit, you will be able to :

- describe the concept of inheritance in C++
- create new classes by reusing the members and properties of existing classes
- describe the advantages and disadvantages of inheritance in pro gramming
- describe how to use base class access specifier *public*, *private* and *protected*
- describe about the use of virtual base class and abstract class

11.2 INTRODUCTION

In this unit, we shall discuss one important and useful feature of Object - Oriented Programming (OOP) which is called ***inheritance***. C++ supports the concept of inheritance. We have already discussed the concept of classes and objects in our previous unit. The knowledge of class and objects is a prerequisite for this unit.

With the help of inheritance we can reuse (or inherit) the property of a previously written class in a new class. There are different types of inheritance which will be discussed in this unit. The concept of abstract and virtual base class will also be covered in this unit.

11.3 INHERITANCE

In Biology, *inheritance* is a term which represents the transformation of the hereditary characters from parents or ancestors to their descendent. In the context of Object-Oriented Programming, the meaning is almost same. The process of creating a new class from existing classes is called ***inheritance***. The newly created class is called ***derived class*** and the existing class is called the ***base class***. The derived class inherits some or all of the characteristics of the base class. Derived class may also possess some other characteristics which are not in the base class.

For example, let us consider two classes namely, “employee ” and “manager”. Whatever information is present in “employee” class, the same will be present in “manager’ also. Apart from that, there will be some extra information in “manager” class due to other responsibilities assinged to managers. Due to the facility of inheritance in C++, it is enough only to indicate those pieces of information which are specific to manager in its class. In addition, the “manager” class will inherit the information of “employee” class.

Before discussing the different types of Inheritance and their implementation in C++, we will first denote the advantages and disadvantages of inheritance.

Advantages of Inheritance

- ***Reusability***

  Reusability is an important feature of Object Oriented Programming. We can reuse code in many situations with the help of inheritance. The base class is defined and once it is compiled, it need not be rewritten. Using the concept of inheritance the programmer can create as many derived classes from the base class as needed. New features can also be added to each derived class when required.

- ***Reliability and Cost***

  Reusability would not only save time but also increase reliability and decrease maintenance cost.

- ***Saves Time and Effort***

  The reuse of class that has already been tested, debugged and used many times can save us the effort of developing and testing the same again.

Disadvantages of Inheritance

- Inappropriate use of inheritance makes a program more complicated.
- In the class hierarchy various data elements remain unused, the memory allocated to them is not utilized.

11.3.1 Defining a Derived Class

A derived class is defined by specifying its relationship with the base class in addition to its own details. The general form of deriving a new class from an existing class is as follows:

```
class DerivedClassName : access specifier  BaseClassName
   {
         .....
         .....    //members of derived class
         .....
   };
```

The derived class name and the base class name are separated by a colon "**:**". We can derive classes using any of the three base class access specifiers: ***public***, ***private*** or ***protected***. If we do not specify the access specifier, then by default it will be *private.* Generally, it is convenient to specify the access specifier while deriving a class. Access specifiers are sometime called *visibility mode*. It determines the access control of the base class members inside the derived class. In the previous unit, we have already learnt about *private* and *public* access specifier while declaring classes. ***Protected*** specifier has a significant role in inheritance.

11.3.2 Accessing Base Class Members

There may be three types of base class derivation:

- Public derivation
- Private derivation
- Protected derivation

- **Public derivation**

When the base class is inherited by using ***public access specifier*** then all public members of the base class become public members of the derived class, and all protected members of the base class become protected members of the derived class. The private members of the base class remain private and are not accessible by members of derived class. Let us examine this with the following example:

// **Program 11.1:** Base class with public access specifier

```
#include<iostream.h>
#include<conio.h>
class base
{
  private:
      int num1,num2;
```

```
  public:
      void input(int n1, int n2 )
                {
                     num1=n1;
                     num2=n2;
                }
      void display( )
      {
          cout<<"Number 1 is:  "<<num1<<endl;
          cout<<"Number 2 is: "<<num2;
      }
  };   //end of base class

class derived : public base
    {
        private:
            int num3;
        public:
            void enter(int n3)
            {
                num3=n3;
            }
            void show()
            {
                cout<<"\nNumber 3 is: "<<num3;
            }
    };  //end of derived class
    int main()
    {
    derived d;     // d is an object of derived class
    clrscr();
    d.enter(15); //enter() function is called by object d
    d.input(5,10);      /* accessing base class member. input()
is a public member of base class */
    d.display(); /* accessing base class member display() is a
```

```
public member of base class */
        d.show();
        getch();
        return 0;
    }
```

OUTPUT:
```
Number 1 is    :  5
Number 2 is    :  10
Number 3 is    :  15
```

Here, ***d*** is a derived class object. With the statement ***d.input(5,15);*** we have made a call to the function *input()* of base class by the derived class object. Similarly, we have called *display( )* member function of base class.

- **Private derivation**

When the base class is inherited by using ***private access specifier***, all the public and protected members of the base class become private members of the derived class. Therefore, public members of base class can only be accessed by the member functions of the derived class and they are not accessible to the objects of the derived class.

For example, if we use the statement

```
class derived : private base
```

instead of `class derived : public base`

in the above program, it will give two error messages while compiling:

Error: base::input(int,int) is not accessible

Error: base::display() is not accessible

As the base class is privately inherited, *input()* and *display()* become private to the derived class althogh they were public in the base class. So other functions like *main( )* cannot access them. Statement like **d.input(5,10);** and **d.display( );** will be invalid in that case.

Protected Members and Inheritance:

Protected members provide greater flexibility in case of inheritance. By using ***protected*** instead of private declaration, we can create class members that are private to their class but that can still be inherited and accessed by derived class. A member declared as protected is accessible by the member functions within its class and any class immediately derived from it.

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 also becomes ready for further inheritance. If a base class is inherited as ***private***, then the ***protected*** member of base class becomes ***private*** in the derived class. Although it is available to the member function of the derived class, it is not available for further inheritance since ***private*** members cannot be inherited.

/***Program 11.2:*** Program showing protected members inherited in public mode */

```
#include<iostream.h>
#include<conio.h>
class base
{
    protected:
int num1,num2;     /*private to base, protected to derived and accessible by derived class member function*/
    public:
    void input(int n1, int n2 )
    {
        num1=n1;
        num2=n2;
    }
    void display( )
    {
```

```
        cout<<"Number 1 is:  "<<num1<<endl;
        cout<<"Number 2 is: "<<num2;
    }
};  //end of base class
class derived : public base//base class is publicly
inherited
{
    private:
    int s;
    public:
    void add( )
    {
    s=num1+num2 ;      /* derived class accessing base class pro-
tected member num1, num2 */
    }
    void show( )
    {
        cout<<"\nSummation is : "<<s;
    }
};  //end of derived class
int main()
{
    derived d;     // d is an object of derived class
    clrscr();
d.input(10,20);  /* accessing base class member. input() is a func-
tion of base class */
d.display(); /* accessing base class member. display() is a function
of base class */
    d.add();
    d.show();
    getch();
    return 0;
}
```

OUTPUT: Number 1 is : 10
Number 2 is : 20
Summation is : 30

The derived class member function *void add()* can access *num1* and *num2* of the base class because *num1* and *num2* are declared as *protected* and the base class access specifier is *public*.

- **Protected derivation**

When the base class is inherited by using ***protected access specifier***, then all protected and public members of base class become protected members of the derived class. Let us consider the following example:

//***Program 11.3*** : Base class derived as protected

```
#include<iostream.h>
#include<conio.h>
class base
{
    protected:
        int num1,num2;
    public:
        void input(int n1, int n2 )
        {
            num1=n1;
            num2=n2;
        }
        void display( )
        {
            cout<<"Number 1 is:  "<<num1<<endl;
            cout<<"Number 2 is: "<<num2;
        }
};  //end of base class

class derived : protected base
{
    private:
```

```
        int s;
    public:
        void add( )
        {
        input(30,60);      /*member function add() of derived
class can access input() as it is inherited as protected */
        s=num1+num2; // num1,num2 are inherited as
                              // protected, so add() can access
            }
            void showall( )
            {
            display();     //display() is inherited as protected
            cout<<"\nSummation is : "<<s;
            }
    };    //end of derived class
    int main()
    {
        derived d;
        clrscr();
    //d.input(10,20); /* invalid. input() is inherited as protected member
of derived. main() can't access it */
    d.add(); /* accessing base class member display() is a function of
base class */
    d.showall(); // public member of derived
    // d.display();     /* invalid. display() can be accessible by derive
class member function only */
        getch();
        return 0;
    }
```

OUTPUT:

```
Number 1 is   :   30
Number 2 is   :   60
Summation is  :   90
```

In the program we can see that *input()*, *display()*, *num1*, *num2* of base class are inherited as protected to derive class. The member functions *add()*, *showall()* of derived class can use them; as protected

members are accessible by derive class members. But main() is not a member function and it cannot access *input()* and *display()*. Although *num1*, *num2* are protected to derived class, but they behave as private to base class.

CHECK YOUR PROGRESS

1. Answer the following by selecting the appropriate option:

   (i) By using protected, one can create class members that
   (a) cannot be inherited and accessed by a derived class
   (b) can be accessed by a derived class
   (c) can be public
   (d) none of these

   (ii) Class members are by default _____
   (a) protected (b) public
   (c) private (d) none of these

   (iii) When base class access specifier is protected, then public members of base class can be accessible by
   (a) member function of derived class
   (b) main() function
   (c) objects of derived class
   (d) none of these

   (iv) Private data members of base class can be inherited by declaring them as
   (a) private (b) public
   (c) protected (d) none of these

   (v) If we donot specify the visibility mode in base class derivation then by default it will be
   (a) protected (b) private
   (c) public (d) none of these

(vi) Private data members can be accessed

(a) from derive class

(b) only from the base class itself

(c) both from the base class and from its derived class

(d) None of these

11.4 TYPES OF INHERITANCE

A program can use one or more base classes to derive a single class. It is also possible that one derived class can be used as base class for another class. Depending on the number of base classes and levels of derivation inheritance is classified into the following forms:

- Single Inheritance
- Multiple Inheritance
- Hierarchical Inheritance
- Multilevel Inheritance
- Hybrid Inheritance
- Multipath Inheritance

11.4.1 Single Inheritance

The programs discussed so far in this unit are examples of single inheritance. In ***single inheritance*** the derived class has only one base class and the derived class is not used as base class. The pictorial representation of single inheritance is given in Fig. 11.1

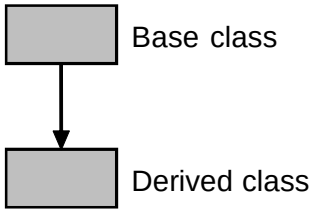

Fig. 11.1: Single Inheritance

The arrow directed from base class towards the derived class indicates that the features of base class are inherited to the derived class. In the following program, we have derived "***employee***" class from "***person***" class. Data member "***name***" and "***age***" are common to both of the two classes. "***name***" and "***age***" are declared as ***protected*** so that derive class can inherit "***name***" and "***age***" from the base class "***person***". Base class is inherited in public mode. Employee may have some other data like designation, salary. So the other data members of "***employee***" class are "***desig***" and "***salary***".

/* **Program 11.4**: Single inheritance with protected data member and public inheritance of base class */

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class person //base class "person"
{
    protected:
    char name[30];    //protected to derived class 'employee'
    float age;
    public:
    void enter(char *nm, float a)/* base class member function */
    {
        strcpy(name,nm);
        age=a;
    }
    void display()    //base class member function
    {
        cout<<"Name: "<<name<<endl;
        cout<<"Age: "<<age<<endl;
    }
};
class employee : public person    */base class "person" is publicly
```

```
                                    inherited */
{                                   */by derived class “employee” */
    private:
    float salary;
    char desig[20];
    public:
    void enter_data(char *n,char *d,float ag,float
s)
    {
    strcpy(name,n);   //”name” of base class can be accessible
                                    //by derived class member function
            strcpy(desig,d);
            salary=s;
            age=ag;   //age can be accessible by
                      //enter_data() of derived class
            }
            void display_all()    //derived class member
function
            {
                display();    //can be used here as publicly
inherited
                cout<<"Designation:
"<<desig<<endl;
                cout<<"Salary: "<<salary<<endl;
            }
};
int main()
{
employee e1,e2;   //e1,e2 are objects of derived class "employee"
    person p;     // p is an object of base class "Person"
    clrscr();
    e1.enter_data("Raktim","Clerk",32,5000);
    cout<<"Employee Details....."<<endl;
```

```
    e1.display_all();
e2.enter("Vaskar",41);      /*erived class object e1 accessing public member of base enter() */
e2.display();       /*derived class object e2 accessing public member of base display() */
    cout<<endl<<"Person Details....."<<endl;
    p.enter("Pragyan",24);
    p.display();
    getch();
    return 0;
}
```

Here, the derived class "***employee***" uses ***name*** and ***age*** of base class "***person***" with the help of derived class member function ***enter_data()***.

Two different classes may have member functions with the same name as well as the same set of arguments. But in case of inheritance, an ambiguous situation arises when base class and derived classes contain member functions with the same name. In main(), if we call member function of that particular name of base class with derived class object, then it will always make a call to the derived class member function of that name. This is because, the function in the derived class *overrides* the inherited function. However, we can invoke the function defined in base class by using the ***scope resolution operator (::)*** to specify the class. For example, let us consider the following program.

/***Program 11.5**: When base and derived class has member functions with same name*/

```
#include<iostream.h>
#include<conio.h>
class B
{
```

```
  protected:
  int p;
  public:
  void enter()
  {
      cout<<"\nEnter an integer:";
      cin>>p;
  }
  void show()
  {
      cout<<"\n\nThe number in Base Class is:
"<<p;
  }
};
class D : public B
{
  private:
  int q,r;
  public:
      void enter() //overrides enter() of "B"
      {
          B::enter();
          cout<<"\nEnter an integer:";
          cin>>q;
      }
      void show()
      {
          r=p*q;
          cout<<"\nEntered numbers in Base and
              Derived class are:"<<p<<"\t"<<q;
          cout<<"\n\nThe product is :"<<r;
      }
};
int main()
{
  D d;          //d is an object of class derived class "D"
```

```
    clrscr();
    d.enter();          //invokes enter() of "D"
    d.show();           //invokes show() of "D"
    d.B::show();  //invokes show() of "B"
    getch();
    return 0;
}
```

In the program, the function name *show()* is same in both base “B” and derived class “D”. To call show() of base class “B”, we have used the statement **d.B::show();** If we use simply **d.show();** then it will invoke show() of derived class “D”.

When a derive class implements a function that has the same name as well as the same set of arguments as the function in the base class, it is called ***function overriding***. When such a function is called through a object of derived class, then the derived class function would be invoked. However, that function in base class would remain hidden.

But there are certain situations where function overriding plays an important role.

11.4.2 Multiple Inheritance

When one class is derived from two or more base classes then it is called ***multiple inheritance***. This type of inheritance allows us to combine the properties of more than one existing classes in a new class. Fig. 11.2 depicts multiple inheritance

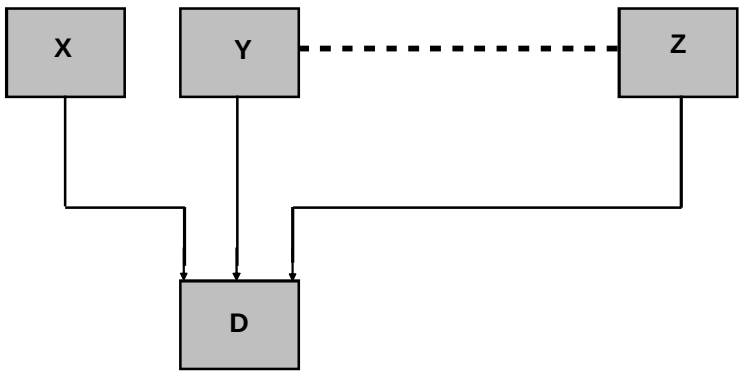

Fig.11.2: Multiple inheritance

We have to specify the base classes one by one separated by commas with their access specifiers. The general form of deriving a derived class from numbers of base class is as follows:

```
class D : public X, public Y, public Z
{
     ......  //body of the derived class
};
```

where X,Y, Z are base classes and D is the derived class. There may be numbers of base classes in multiple inheritance which is indicated by the dotted line in the figure.

For demonstration of multiple inheritance let us consider the following program. There are three base classes and one derived class. The derived class CHARACTER has one private member “n” and two public member functions “enter()” and “show()”. The function “enter()” is used to read a number, a vowel, a consonent and a symbol from the keyboard and the “show()” function is used to display the contents on the screen. The class members of all the three base classes are publicly derived.

// **Program 11.6** : Example of Multiple inheritance

```
#include<iostream.h>
#include<conio.h>
class V //base class
{
    protected:
        char v;
};
class C //base class
{
    protected:
        char c;
};
class S //base class
```

```
{
    protected:
        char s;
};
class CHARACTER : public V, public C, public S
{
    private:
        int n;
    public:
        void enter() //derived class member function
        {
            cout<<"\nEnter a vowel:";
        cin>>v;//accessing protected member v of class
"VOWEL"
            cout<<"\nEnter a consonent:";
        cin>>c;  //accessing c of "CONSONENT" class
        cout<<"\nEnter a symbol:";
        cin>>s   //accessing s of "SYMBOL" class
        cout<<"\nEnter a number:";
        cin>>n;  //accessing n of "NUMBER" class
    }
    void show()
    {   cout<<"\nThe entered characters are
:\n\n";
        cout<<"\nVowel: "<<v;
        cout<<"\nConsonent: "<<c;
        cout<<"\nSymbol: "<<s;
        cout<<"\nNumber: "<<n;
    }
};
int main()
{
    CHARACTER o; //o is an object of derived class
"character"
    clrscr();
```

```
    o.enter();
    o.show();
    getch();
    return 0;
}
```

One suitable example of the implementation of multiple inheritance is shown in the program below:

/***Program 11.7***: Program showing multiple inheritance with two base class (practical, theory) and one derived class (result) /*

```
#include<iostream.h>
#include<conio.h>
class practical  //base class "practical"
{
    protected:
    float p1_marks, p2_marks,total;
    public:
    void practical_marks()
    {
    cout<<"Enter marks of practical paper 1 and
paper 2: ";
        cin>>p1_marks>> p2_marks;
    }
    float add()  //returns the total of practical
    {
        total=p1_marks+p2_marks;
        return total;
    }
    void display_practical()
    {
        cout<<endl<<"Total     Practical
marks:"<<total;
    }
};      //end of "practical" class
```

```
class theory // base class “theory”
{
    protected:
    float phy, chem, math, total_marks;
    public:
    void theory_marks(){
cout<<"Enter marks of Physics, Chem and Maths:";
    cin>>phy>>chem>>math;
}
float sum()
{
    total_marks=phy+chem+math;
    return total_marks;
}
void display_theory()
{
    cout<<endl<<"Total              Theory
marks:"<<total_marks;
}
};  //end of base class “theory”
class result : public practical,public theory
{
    protected:
    int roll;
    float grand_total,t,p;
    public:
    void enter() {
cout<<"ENTER STUDENT INFORMATION......"<<endl;
    cout<<"Enter Roll no.:";
    cin>>roll;
theory_marks();  //inherited publicly from base class "theory"
practical_marks();    //inherited from base class "practical"
}
void theory_practical()
{
    t=sum()
```

```
    p=add();
    grand_total=t+p;
cout<<"\nThe total marks of the student is:"
<<grand_total;
}
};  //end of derived class "result"
int main()
{
    result s1;   //object of derived class
    clrscr();
    s1.enter();
    s1.theory_practical();// accessing derived class mem-
ber
s1.display_theory(); //s1 accessing "display_theory()" of
"theory"
    s1.display_practical();
    getch();
    return 0;
}
```

When we execute the program entering marks for practical and theory papers for a particular student, then it will display the result as follows:

ENTER STUDENT INFORMATION.....
Enter Roll no.: 1
Enter marks of Physics, Chemistry and Mathematics: 65 72 81
Enter marks of practical paper 1 and paper 2 : 25 26
The total marks of the student is : 269
Total Theory marks : 218
Total Practical marks : 51

The above program consists of three classes: two base classes ("***practical***" and "***theory***") and one derived class ("***result***"). The member function "***enter()***" of derive class inherits member functions "***practical_mark()***" and "***theory_ marks()***" of base class "***practical***"

and "***theory***" respectively. Similarly, member function "***theory_practical()***" uses "***sum()***" and "***add()***" of base class to calculate the "*grand_total*" marks of student. Thus, in the derived class we need not have to write functions for entering practical and theory marks. We just inherit them from the base classes.

EXERCISE

Q. Suppose a class D is derived from class B. B has two public member functions *getdata()* and *showdata()* and D has two public functions *readdata()* and *displayall()*. Define the classes such that both function *getdata()* and *showdata()* should be accessible in the *main()* function.

11.4.3 Hierarchical Inheritance

Derivation of more than one classes from a single base class is termed as ***hierachical inheritance***. This is a very common form of inheritance in practice. The rules for defining such classes are the same as in single inheritance. The pictorial representation of hierarchical inheritance is shown in Fig. 11.3

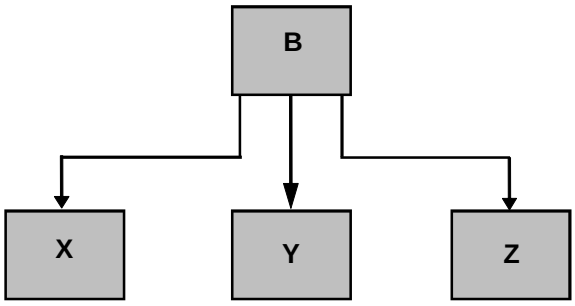

Figure11 3: Hierarchical Inheritance

For demonstration of hierarchical inheritence let us consider a program with one base class ("student") and three derived classes ("arts", "science" and "commerce").

//**Program 11.8**: Demonstration of hierarchical inheritance

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

```
#include<conio.h>
class student    // base class  “student”
{
    protected:
    char fname[20],lname[20];
    int age,roll;
    public:
    void student_info()
    {
        cout<<"Enter the first name and last name:
";
        cin>>fname>>lname;
        cout<<"\nEnter the Roll no.and Age: ";
        cin>>roll>>age ;
    }
    void display()
    {
        cout<<"\nRoll Number = "<<roll;
        cout<<"\nFirst        Name        =
"<<fname<<"\t"<<lname;
        cout <<"\nAge = " << age;
    }
};
class arts : public student   //derived class arts
{
    private:
    char asub1[20], asub2[20], asub3[20] ;
    public:
    void enter_arts()
    {
        student_info(); //base class member function
        cout <<"\n Enter the subject1 of the arts
student:";
    cin >> asub1 ;
    cout<<"\nEnter the subject2 of the arts stu-
dent:";
```

```
    cin >> asub2 ;
    cout<<"\nEnter the subject3 of the arts stu-
dent:";
    cin >> asub3 ;
}
void display_arts()
{
    display();//base class member function
    cout<<"\n\t Subject1 of the arts student="<<
asub1;
    cout<<"\n\t Subject2 of the arts student="<<
asub2;
    cout<<"\n\t Subject3 of the arts student="<<
asub3;
}
};
class commerce : public student    //derived class
"commerce"
{
    private:
    char csub1[20], csub2[20], csub3[20] ;
    public:
    void enter_com(void)
    {
    student_info();  //base class member function
cout<<"\tEnter the subject1 of the commerce student:";
    cin>> csub1;
cout<<"\tEnter the subject2 of the commerce student:";
    cin>>csub2 ;
cout<<"\tEnter the subject3 of the commerce student:";
    cin>> csub3 ;
    }
    void display_com()
    {
    display();   //base class member function
cout<<"\nSubject1 of the commerce student="
```

```
<<csub1;
cout<<"\nSubject2 of the commerce student="
<<csub2;
cout<<"\nSubject3 of the commerce student="<< csub3
    }
    };
class science : public student  //derived class "science"
{
    private:
    char ssub1[20], ssub2[20], ssub3[20] ;
    public:
    void enter_sc(void)
    {
    student_info();  //base class member function
cout<<"\nEnter the subject1 of science student:";
    cin>>ssub1;
cout<<"\nEnter the subject2 of science student:";
    cin>>ssub2 ;
cout<<"\nEnter subject3 of the science student:";
    cin>>ssub3 ;
    }
    void display_sc()
    {
    display();   //base class member function
cout<<"\nSubject1 of the science student="<< ssub1
cout<<"\nSubject2 of the science student="<< ssub2;
cout<<"\nSubject3 of the science student="<< ssub3;
    }
};
int main()
{
    arts a ; //a is an object of derived class "arts"
    clrscr();
    cout << "\n Entering details of the arts
student\n";
```

```
    a.enter_arts( );
    cout <<"\nDisplaying the details of arts student\n";
    a.display_arts( );
    science s;   //s is an object of derived class "science"
    cout <<"\n\n Entering details of science student\n"
    s.enter_sc( );
    cout<<"\n Display details of the science student\n";
    s.display_sc( );
    commerce c ;    //c is an object of derived class "commerce"
    cout<<"\n\nEnter details of commerce student\n";
    c.enter_com( );
    cout << "\n Display details of commerce student\n";
    c.display_com() ;
    getch();
    return 0;
}
```

11.4.4 Multilevel Inheritance

C++ also provides the facility of ***multilevel inheritance***, according to which the derived class can also be derived by an another class, which in turn can further be inherited by another and so on. For instance, a class X serves as a base class for class Y which in turn serves as base class for another class Z. The class Y which forms the link between the classes X and Y is known as the ***intermediate base class***. Further, Z can also be used as a base class for another new class. The following figure depicts multilevel inheritance.

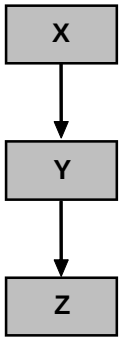

Figure11.4: Multilevel Inheritance

11.4.5 Hybrid Inheritance

It is possible to derive a class involving more than one type of inheritance. ***Hybrid inheritance*** is that type of inheritance where several forms of inheritance are used to derive a class. There could be situations where we need to apply two or more types of inheritance to design a particular program.

For example, let us assume that we are to design a program which will select players for a particular compitetion. For this purpose we could consider four classes PLAYER, GAME, RESULT and PHYSIQUE. PLAYER class contains the player details including name, address, location etc. GAME class can be derived from PLAYER class. Again, if weightage for physical test should be added before finalizing the result, then we can inherit that from PHYSIQUE. RESULT class is derived from two base classes GAME and PHYSIQUE. The following diagram gives us the inhertance relationship between various classes.

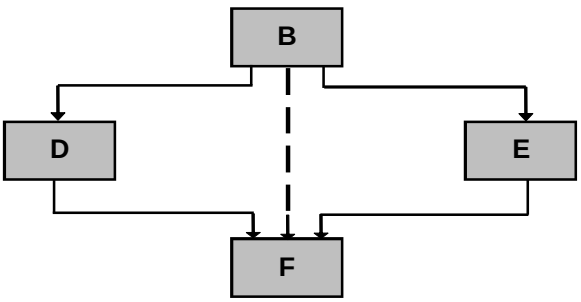

Fig.11.5: Hybrid Inheritance

In the diagram, RESULT has two base classes,GAME and PHYSIQUE. GAME is not only a base class but also a derived class. Here we can see that two types of inheritance *multiple* and *multilevel* are combined to create the RESULT class.

11.4.6 Multipath Inheritance

The inheritance where a class is derived from two or more classes, which are in turn derived from the same base class is known as ***multipath inheritance***. There may be many types of inheritance such as multiple, multilevel, hierarchical etc. in multipath inheritance. Certain difficulties may arise in this type of inheritance. Suppose we have two derived classes D and E that have a common base class B, and we have another class F that inherits from D and E.

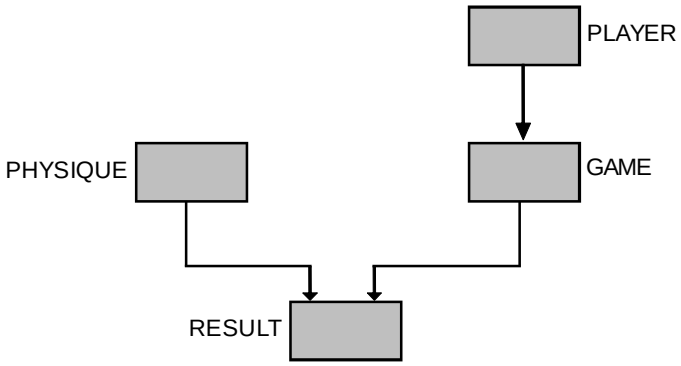

Fig.11.6 : Multipath Inheritance

In the above diagram, we can observe three types of inheritances, i.e., multiple, multilevel and hierarchical. For better illustration let us consider the following program.

// **Program 11.9:** Demosntration of multipath inheritance

```
#include<iostream.h>
#include<conio.h>
class B
{
    protected:
    int b;
```

```
};
class D : public B   //B is publicly inherited by D
{
    protected:
    int d;
};
class E : public B   //B is publicly inherited by E
{
    protected:
    int e;
};
class F : public D, public E//D,E are publicly inherited
by F
{
    protected:
    int f;
    public:
    void enter_number()
    {
        cout<<"Enter some integer values for
b,d,e,f:";
        cin>>b>>d>>e>>f;
    }
    void display()
    {
    cout<<"\nEntered numbers are:\n";
    cout<<"\nb= "<<b<<"\nd= "<<d<<"\ne= "<<e<<"\nf=
"<<f;
    }
};
int main()
{
    F obj;       //instantiaton of class F
    clrscr();
    obj.enter_number();  //enter_number() of F is called
```

```
    obj.display();
    getch();
    return 0;
}
```

Now, if we instantiate class F and call the functions ***enter_number()*** and ***display()***, then the compiler shows the following types error messages:

Error M.cpp 28: Member is ambiguous:'B ::b' and 'B::b'

Error M.cpp 33: Member is ambiguous:'B ::b' and 'B::b'

This is due to the duplication of members of class B in F. The member **b** of class B is inherited twice to class F: one through class D and another through class E. This leads to ambiguity. To avoid such type of situation, ***virtual base class*** is introduced.

11.5 VIRTUAL BASE CLASSES

C++ provides the concept of ***virtual base class*** to overcome the ambiguity occurring due to *multipath inheritance*. While discussing multipath inheritance, we have faced a situation which may lead to duplication of inherited members in the derived class F (Fig.11.6). This can be avoided by making the common base class (i.e.,B) as *virtual base class* while derivation. We can declare the base class B as ***virtual*** to ensure that D and E share the same data member B.

This is shown in the following program which is the modification of the previous program 11.9.

/***Program 11.10:** Virtual base class and removal of ambiguity occuerd in multipath inheritance */

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

```
{
    protected:
    int b;
};
class D:public virtual B
//B is publicly inherited by D and made virtual
{
    protected:
    int d;
};
class E:public virtual B
//B is publicly inherited by E and made virtual
{
    protected:
    int e;
};
class F : public D, public E
//D,E are publicly inherited by F
{
    protected:
    int f;
    public:
    void enter_number()
    {
        cout<<"Enter some integer values for b,d,e,f:
";
        cin>>b>>d>>e>>f;
    }
    void display()
    {
        cout<<"\nEntered numbers are:\n";
        cout<<"\nb= "<<b<<"\nd= "<<d<<"\ne=
"<<e<<"\nf= "<<f;
```

```
    }
};
int main()
{
    F obj;  //instantiaton of class F
    clrscr();
    obj.enter_number();
    obj.display();
    getch();
    return 0;
}
```

Here we have used the keyword virtual in front of the base class specifiers to indicate that only one subobject of type B, shared by class D and class E, exists.. When a class is made a ***virtual base class***, C++ takes the necessary action that only one copy of that class is inherited, regardless of how many paths exist between the virtual base class and a derived class.

11.6 ABSTRACT CLASSES

The objects created often are the instances of a derived class but not of the base class. The base class just becomes a structure or foundaion with the help of which other classes are built and hence such classes are called ***abstract class*** or ***abstract base class***. In other words, when a class is not used for creating objects then it is called *abstract class*. In the *Program 10.10*, B is an abstract class since it was not used for creating any object.

LET US KNOW

Inheritance and Constructors, Destructors

Although constructors are suitable for initializing objects , we have not used them in any program in this unit for the sake of simplicity. But if we use contructors in program, then we must follow certain definite rules

while inheriting derive classes. If the base class contains no argument constructor then the derived class does not require a constructor. If any base class contains parameterized constructor, then it is mandatory for the derived class to have a constructor and pass the arguments to the base class constructors. In case of inheritance, normally derived classes are used to declare objects. Hence it is necessary to define constructor in the derived class. When both the derived and base classes contain constructors, the base constructor is executed first and then the constructor in derived class is executed. Destructors are executed in the reverse order of constructor execution.

CHECK YOUR PROGRESS

2. Select whether the following statements are True (T) or False (F):
   (i) A class can serve as base class for many derived classes.
   (ii) When one class is derived from another derived class then that is called *multiple inheritance*.
   (iii) When one class is derived from more than one base class then that is called *multiple inheritence*.
   (iv) When more than one form of inheritance is used in designing a class then that type is called *hybrid inheritance*.
3. Answer the following by selecting the appropriate option:
   (i) In multilevel inheritance, the middle class acts as
   (a) only derived class (b) only base class
   (c) base class as well as derived class
   (d) none of the above
   (ii) A class is declared “virtual” when
   (a) more than one class is derived
   (b) two or more classes have common base class
   (c) there are more than one base classes

(d) none of the above

(iii) When a class is not used for creating objects, it is called

(a) abstract class (b) virtual base class

(c) derived class (d) none of these

(iv) *Intermediate base class* is present in case of

(a) single inheritance (b) multiple inheritance

(c) multilevel inheritance (d) hierarchical inheritance

11.7 LET US SUM UP

- ***Inheritance*** is one of the most useful and essential characteristics of object-oriented programming language. If we have developed a class and we want a new class that is almost similar, but slightly different, the principles of inheritance come handy. The existing class is known as ***base*** class and the newly formed class is known as ***derived*** class.The derived class can have some other characteristics which are not in base class.
- Private members of a class cannot be inherited either in public mode or in private mode.
- When a public member inherited in public, protected and private mode, then in derived class it remains with the same access specifiers as in base class i.e., public, protected and private respectively.
- A protected member inherited in public mode becomes protected, whereas inherited in private mode becomes private in derived class.
- The protected and public data
- In ***single inheritance***, one new class is derived from a single base class.
- When a class is derived using the properties of several base classes, then it is called ***multiple inheritance.***

- The process of deriving a class from another derived class is called ***multilevel inheritance***.
- More than one class can be derived from only one base class i.e., characteristics of one class can be inherited by more than one class. This is called ***hierarchical inheritance***.
- When different types of inheritance are applied in a single program then it is termed as ***hybrid inheritance***.
- When a class is derived from two or more classes, which are derived from the same base class, such type of inheritance is known as ***multipath inheritance***.
- We can make a class ***virtual*** if it is a base class that has been used by more than one derived class as their base class. When classes are declared as *virtual*, the compiler takes necessary caution to avoid the duplication of the member variables.
- When a class is not used for creating objects, it is called an ***abstract class***.

11.8 FURTHER READING

- Balagurusamy, E. (2011), *Object-oriented programming with C++*, 6e. Tata McGraw-Hill Education
- Venugopal, K.R. (2013), Rajkumar, *Mastering C++*. Tata McGraw-Hill Education
- Ravichandan D. (2002), *Programming with C++*, 2e. Tata McGraw-Hill Education

11.9 ANSWERS TO CHECK YOUR PROGRESS

1. (i) (b) can be accessed by a derived class
   (ii) (c) private
   (iii) (a) member function of derived class
   (iv) (c) protected

(v) (b) private

(vi) (b) only from the base class itself

2. (i) True (ii) False
   (iii) True (iv) True

3. (i) (c) base class as well as derived class
   (ii) (b) two or more classes have common base class
   (iii) (a) abstract class
   (iv) (c) multilevel inheritance

11.10 MODEL QUESTIONS

1. What does inheritance mean in C++?
2. What are the types of inheritance? Explain any three of them with examples.
3. What are the different types of visibility modes of base class?
4. Write a program to derive a class from multiple base classes.
5. When do we make a class virtual?
6. What are abstract base classes?
7. Explain multipath inheritance.
8. Write a C++ program involving appropriate type of inheritance which will inherit two classes *triangle* and *rectangle* from *polygon* class. Use member functions for entering appropriate parameters like *width*,*height* etc. and to calculate the area of triangle and rectangle.
9. Consider a case of University having the disciplines of Engineering, Management, Science, Arts and Commerce. There are many colleges in the University. Assuming a college can run a course partaining to only one discipline, draw the class diagram.To which type of inheritance does this structure belong?

****