Inheritance

Learning Objectives

After the completion of this chapter, the student will be able to

- Understand the purpose of Inheritance
- Construct C++ programs using Inheritance
- Execute and debug programs which contains the concept of Inheritance

16.1 Introduction to Inheritance

Inheritance is one of the most important features of Object Oriented Programming. In object-oriented programming, inheritance enables new class and its objects to take on the properties of the existing classes. A class that is used as the basis for creating a new class is called a superclass or base class. A class that inherits from a superclass is called a subclass or derived class

16.2 Need for Inheritance

Inheritance is an important feature of object oriented programming used for code reusability. It is a process of creating new classes called derived classes, from the existing or base classes. Inheritance allows us to inherit all the code (except declared as private) of one class to another class. The class to be inherited is called base class or parent class and the class which inherits the other class is called derived class or child class. The derived class is a power packed class, as it can add additional attributes and methods and thus enhance its functionality.

Notes

The main advantage of inheritance is

- It represents real world relationships well
- It provides reusability of code
- It supports transitivity

16.3 Types of Inheritance

There are different types of inheritance viz., Single Inheritance, Multiple inheritance, Multilevel inheritance, hybrid inheritance and hierarchical inheritance.

1. Single Inheritance

When a derived class inherits only from one base class, it is known as single inheritance

2. Multiple Inheritance

When a derived class inherits from multiple base classes it is known as multiple inheritance

3. Hierarchical inheritance

When more than one derived classes are created from a single base class , it is known as Hierarchical inheritance.

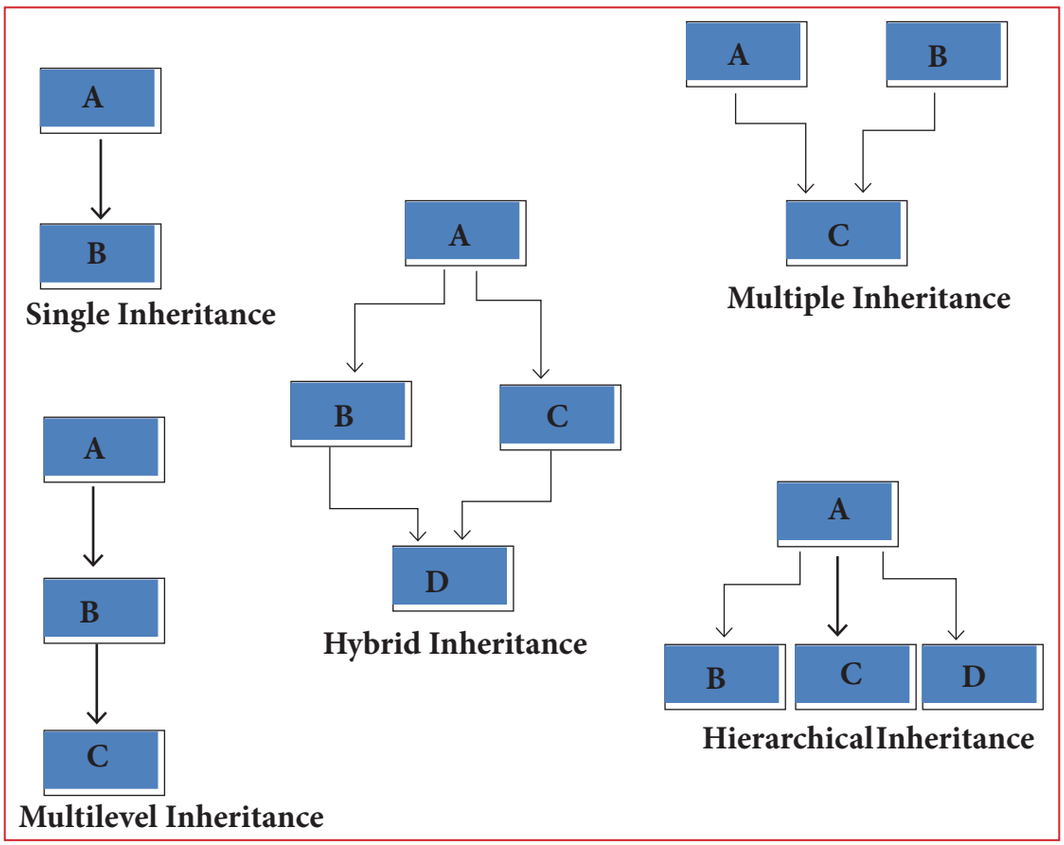
4. Multilevel Inheritance

The transitive nature of inheritance is reflected by this form of inheritance. When a class is derived from a class which is a derived class – then it is referred to as multilevel inheritance.

5. Hybrid inheritance

When there is a combination of more than one type of inheritance, it is known as hybrid inheritance. Hence, it may be a combination of Multilevel and Multiple inheritance or Hierarchical and Multilevel inheritance or Hierarchical, Multilevel and Multiple inheritance.

The following diagram represents the different types of inheritance



16.4 Derived Class and Base class

While defining a derived class, the derived class should identify the class from which it is derived.The following points should be observed for defining the derived class.

i The keyword class has to be used

ii The name of the derived class is to be given after the keyword class

iii A single colon (:)

iv The type of derivation (the visibility mode), namely private, public or protected. If no visibility mode is specified ,then by default the visibility mode is considered as private.

v The name of the base class(parent class), if more than one base class, then it can be given separated by comma.

class derived_class_name :visibility_mode base_class_name

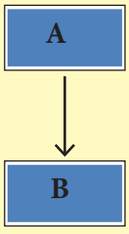
{ // members of derivedclass };

The following are some of the examples for different forms of inheritance

16.4.1 Single Inheritance

Illustration 16.1 Single inheritance

```
# include <iostream>
using namespace std;
class student          //base class
{
private :
char name[20];
int rno;
public:
void acceptname()
{
cout<<"\n Enter roll no and name .. ";
cin>>rno>>name;
}
void displayname()
{
cout<<"\n Roll no :-"<<rno;
cout<<"\n Name :-"<<name<<endl;
}
};
class exam : public student          //derived class with single base class
 {
public:
          int  mark1, mark2 ,mark3,mark4,mark5,mark6,total;
  void acceptmark()
  {
    cout<<"\n Enter lang,eng,phy,che,csc,mat marks.. ";
    cin>>mark1>>mark2>>mark3>>mark4>>mark5>>mark6;
}
void displaymark()
{
cout<<"\n\t\t Marks Obtained ";
cout<<"\n Language.. "<<mark1;
cout<<"\n English .. "<<mark2;
cout<<"\n Physics .. "<<mark3;
cout<<"\n Chemistry.. "<<mark4;
cout<<"\n Comp.sci.. "<<mark5;
cout<<"\n Maths ..   "<<mark6;
cout<<"\n Total ..   "<<mark1+mark2+mark3+mark4+mark5+mark6;
}
 };
int main()
```



Single inheritance

```
int main()
{
  exam e1;
 e1.acceptname();      //calling base class function using derived class object
 e1.acceptmark();
 e1.displayname();       //calling base class function using derived class object
 e1.displaymark();
 return 0;
}
```

Output:

```
Enter roll no and name .. 1201 KANNAN
Enter lang,eng,phy,che,csc,mat marks.. 95 90 92 96 97 99
Roll no :-1201
Name :-KANNAN
                    Marks Obtained
Language.. 95
English .. 90
Physics .. 92
Chemistry.. 96
Comp.sci.. 97
Maths ..   99
Total ..   569
```

In the above program the derived class "exam" inherits all the members of the base class "student". But it has access privilege only to the non private members of the base class.

16.4.3 Multilevel Inheritance

Illustration 16.2 Multilevelw inheritance

```
# include <iostream>
using namespace std;
class student              //base class
{
private :
char name[20];
int rno;
public:
void acceptname()
{ cout<<"\n Enter roll no and name .. "; cin>>rno>>name;
}
void displayname()
{ cout<<"\n Roll no :-"<<rno;
cout<<"\n Name :-"<<name<<endl;   }};
 class exam : public student          //derived class with single base class
```

```
{
        public:
        int  mark1, mark2 ,mark3;
        void acceptmark()
        {       cout<<"\n Enter 3 subject marks.. ";
                cin>>mark1>>mark2>>mark3;     }
void displaymark(){
cout<<"\n\t\t Marks Obtained ";
cout<<"\n Subject1... "<<mark1;
cout<<"\n Subject2... "<<mark2;
cout<<"\n Subject3... "<<mark3;   }   };
class result : public exam
{
int total;
public:
void showresult()
{
total=mark1+mark2+mark3;
cout<<"\nTOTAL MARK SCORED : "<<total;
}
};
int main()
{
        result r1;
r1.acceptname();          //calling base class function using derived class object
r1.acceptmark();          //calling base class function which itself is a derived
                          // class  function using its derived class object
r1.displayname();         //calling base class function using derived class object
r1.displaymark();         /*calling base class function which itself is a derived
                          class  function using its derived class object*/
r1.showresult();          //calling the child class function
return 0;
}
```

Output:

```
Enter roll no and name .. 1201 Lohit Sarathi
 Enter 3 subject marks.. 98 100 100
 Roll no :-1201
 Name :-Lohith Sarathi
            Marks Obtained
Subject1 ...  98
Subject 2 ...  100
Subject 3... 100
TOTAL MARK SCORED    :  298
```

In the above program class "result " is derived from class "exam" which itself is derived from class student.

Note

In multilevel inheritance the level of inheritance can be extended to any number of level depending upon the relation. Multilevel inheritance is similar to relation between grandfather, father and child.

A class without any declaration will have 1 byte size.class x{}; X occupies 1 byte.

In the above program the derived class "result" has acquired the properties of class "detail" and class "exam" which is derived from "student". So this inheritance is a combination of multi level and multiple inheritance and so it is called hybrid inheritance

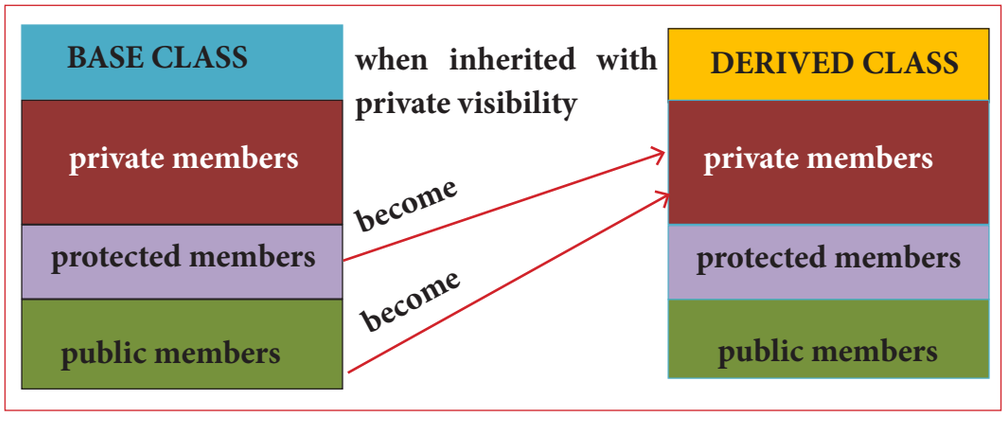
16.5 VISIBILITY MODES

An important feature of Inheritance is to know which member of the base class will be acquired by the derived class. This is done by using visibility modes.

The accessibility of base class by the derived class is controlled by visibility modes. The three visibility modes are private, protected and public. The default visibility mode is private. Though visibility modes and access specifiers look similar, the main difference between them is Access specifiers control the accessibility of the members with in the class where as visibility modes control the access of inherited members with in the class.

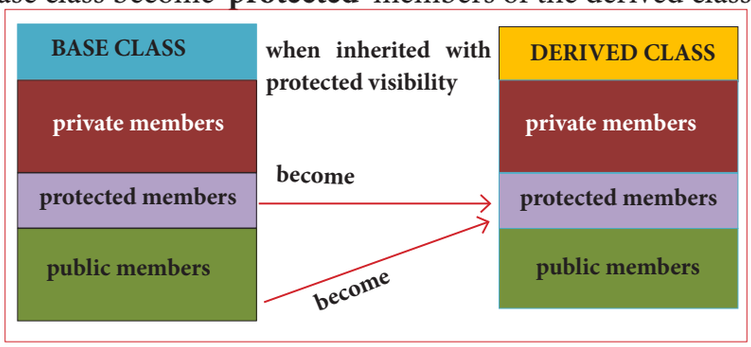
16.5.1 Private visibility mode

When a base class is inherited with **private** visibility mode the public and protected members of the base class become **'private'** members of the derived class



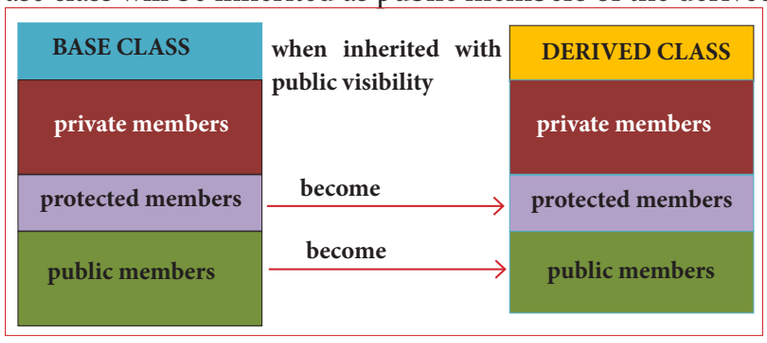
16.5.2 protected visibility mode

When a base class is inherited with **protected** visibility mode the protected and public members of the base class become '**protected**' members of the derived class



16.5.3 public visibility mode

When a base class is inherited with **public** visibility mode , the protected members of the base class will be inherited as **protected** members of the derived class and the **public** members of the base class will be inherited as public members of the derived class.



Tip Notes

When classes are inherited with public, protected or private the private members of the base class are not inherited they are only visible i.e continue to exist in derived classes, and cannot be accessed

Illustration 16.3 Explains the significance of different visibility modes.

```
//Implementation of Single Inheritance using public visibility mode
#include <iostream>
using namespace std;
class Shape
{    private:
          int count;
   protected:
          int width, height;
  public:
          void setWidth(int w)
          {          width = w;      }
void setHeight(int h)
{ height = h; } };
```

```
class Rectangle: public Shape
{
public:
int getArea()
{
return (width * height);
}
};
int main()
{
Rectangle Rect;
Rect.setWidth(5);
Rect.setHeight(7);
// Print the area of theobject.
cout<< "Total area: "<<Rect.getArea() <<endl;
return 0;
}
```

Output
Total area: 35

The following table contain the members defined inside each class before inheritance

MEMBERS of class	visibility modes		
	Private	Protected	Public
Shape(base class)	int count;	int width; int height;	void setWidth(int) void setHeight(int)
Rectangle (derived class only with its defined members)			int getArea();

The following table contains the details of members defined after inheritance

MEMBERS of class	visibility modes –public for acquiring the properties of the base class		
	Private	Protected	Public
Shape(base class)	int count;	int width; int height;	void setWidth(int) void setHeight(int)
Rectangle (derived class acquired the properties of base class with public visibility)	Private members of base classes are not directly accessible by the derived class	int width; int height;	int getArea(); void setWidth(int) void setHeight(int)

Suppose the class **rectangle is derived with protected visibility** then the properties of class rectangle will change as follows

MEMBERS of class	visibility modes –protected for acquiring the properties of the base class		
	Private	Protected	Public
Shape(base class)	int count;	int width; int height;	void setWidth(int) void setHeight(int)
Rectangle (derived class acquired the properties of base class with protected visibility)	Private members of base classes are not directly accessible by the derived class	int width; int height; void setWidth(int) void setHeight(int)	int getArea();

In case the class rectangle is derived with private visibility mode from its base class shape then the property of class rectangle will change as follows

MEMBERS of class	visibility modes –private for acquiring the properties of the base class		
	Private	Protected	Public
Shape(base class)	int count;		void setWidth(int) void setHeight(int)
Rectangle (derived class acquired the properties of base class with private visibility)	int width; int height; void setWidth(int) void setHeight(int)		int getArea();

When you derive a class from an existing base class,it may inherit the properties of the base class based on its visibility mode.So one must give appropriate visibility mode depending on the need.

Private inheritance should be used when you want the features of the base class to be available to the derived class but not to the classes that are derived from the derived class.

Protected inheritance should be used when features of base class to be available only to the derived class members but not to the outside world.

Public inheritance can be used when features of base class to be available to the derived class members and also to the outside world.

16.6 Inheritance and constructors and destructors

When an object of the derived class is created ,the compiler first call the base class constructor and then the constructor of the derived class. This is because the derived class is built up on the members of the base class. When the object of a derived class expires first the derived class destructor is invoked followed by the base class destructor.

Illustration 16.4 The order of constructors and destructors

```
#include<iostream>
using namespace std;
class base
{
public:
base()
{ cout<<"\nConstructor of base class..."; }
~base()
{ cout<<"\nDestructor of base class.... "; }
};
class derived:public base
{
public :
derived()
{ cout << "\nConstructor of derived ..."; }
~derived()
{ cout << "\nDestructor of derived ..."; }};
class derived1 :public derived
{
public :
derived1()
{ cout << "\nConstructor of derived1 ...";}
~derived1()
{ cout << "\nDestructor of derived1 ...";}
};
int main()
{
derived1 x;
return 0;
}
```

Output:

```
Constructor of base class...
Constructor of derived ...
Constructor of derived1 ...
Destructor of derived1 ...
Destructor of derived ...
Destructor of base class....
```

Notes

- The constructors are executed in the order of inherited class i.e., from base constructor to derived. The destructors are executed in the reverse order.
- The size of derived class object=size of all base class data members + size of all data members in derived class.

16.7 Overriding / Shadowing Base class functions in derived class

In case of inheritance there are situations where the member function of the base class and derived classes have the same name. If the derived class object calls the overloaded

member function it leads to confusion to the compiler as to which function is to be invoked. The derived class member function have higher priority than the base class member function. This shadows the member function of the base class which has the same name like the member function of the derived class. The scope resolution (::) operator resolves this problem.

Illustration 16.5 The use of scope resolution operator in derived class

```
#include<iostream>
#include<string>
using namespace std;
class Employee
{
   private:
      char name[50];
      int code;
   public:
      void getdata();
      void display();
};
class staff: public Employee
{
   private:
      int ex;
   public:
         void getdata();
      void display();
};
void Employee::display()
{
         cout<<"Name:"<<name<<endl;
         cout<<"Code:"<<code<<endl;
}
void staff::display()
{        Employee :: display();//overriding
         cout<<"Experience:"<<ex<<" Years"<<endl;
}
void Employee::getdata()
{
cout<<"Name:";
cin>>name;
cout<<"\ncode:";
cin>>code;
}
void staff::getdata()
{
Employee::getdata();
cout<<"\nExperience:";
cin>>ex;
int main()
{
         staff s;
         cout<<"Enter data"<<endl;
         s.getdata();
         cout<<endl<<endl<<"\tDisplay Data"<<endl;
         s.display();
         return 0;
}
```

Output

```
Enter data
Name: SUGANYA
Code: 1201
Experience: 30
        Display Data
Name: SUGANYA
Code:1201
Experience:30 Years
```

In the above program getdata() and display() are defined both in base and in derived class. So when the derived class staff inherits the properties of Employee class it will have two getdata() and display() each. To differentiate the derived getdata() and display() from the defined getdata() and display() :: (scope resolution) operator is given along with the base class name to the base class members

Note

When a derived class member function has the same name as that of its base class member function ,the derived class member function shadows/hides the base class's inherited function .This situation is called function overriding and this can be resolved by giving the base class name followed by :: and the member function name.

Points to Remember:

- The mechanism of deriving new class from an existing class is called inheritance.
- The main advantage of Inheritance is it supports reusability of code.
- The derived class inherits all the properties of the base class. It is a power packed class, as it can add additional attributes and methods and thus enhance its functionality.
- The various types of Inheritance are Single inheritance, multiple inheritance, multilevel inheritance, hierarchical inheritance and hybrid inheritance
- When a derived class inherits only from one base class, it is known as single inheritance
- When a derived class inherits from multiple base classes itis known as multiple inheritance
- When a class is derived from a class which is a derived class itself – then this is referred to as multilevel inheritance. The transitive nature of inheritance is reflected by this form of inheritance.
- When more than one derived classes are created from a single base class , it is known as Hierarchical inheritance.
- When there is a combination of more than one type of inheritance, it is known as hybrid inheritance.
- In multiple inheritance, the base classes are constructed in the order in which they appear in the declaration of the derived class.
- A sub-class can derive itself publicly, privately or protectedly.
- The private member of a class cannot be inherited .
- In publicly derived class,the public members of the base class remain public and protected members of base class remain protected in derived class.
- In privately derived class, the public and the protected members of the base class become private in derived class
- When class is derived in protected mode, the public and protected members of base class become protected in derived class.
- constructors and destructors of the base class are not inherited but during the creation of an object for derived class the constructors of base class will automatically be invoked.
- The destructors are invoked in reverse order .The destructors of the derived classes are invoked first and then the base class.
- The size of derived class object=size of all base class data members + size of all data members in derived class
- overriding of the members are resolved by using Scope resolution operator(::).
- this pointer is used to refer to the current objects members