



# Chapter 10

## Inheritance

### **Learning Objectives**

**After completing this chapter, you will be able to:**

- *Understand inheritance.*
- *Understand single inheritance.*
- *Understand multilevel inheritance.*
- *Understand multiple inheritance.*
- *Understand hierarchical inheritance.*
- *Understand hybrid inheritance.*
- *Understand virtual base classes.*

## INTRODUCTION

In this chapter, you will learn about another feature of OOP that is known as reusability. Reusability provides a mechanism with which you can reuse something that already exists. This mechanism saves time because you do not need to create the same thing again and again. In C++, reusability is used in the case of classes. Once a class has been written and debugged, it can be used in several ways. In C++, reusability is used by creating a new class from an existing class by reusing its properties. The technique of creating a new class from an old one is known as inheritance or derivation. The new class is known as derived class or subclass and the old class (which is inherited) is known as base class. A derived class can inherit some or all the properties of a base class. A derived class can also add some properties on its own. The syntax for defining a derived class is as follows:

```
class derived_class_name : access_specifier base_class_name
{
    data members;
    access_specifier:
        member functions;
};
```

In the above syntax, the name of the derived class is represented by **derived\_class\_name**. Here, the colon (:) specifies that the derived class represented by **derived\_class\_name** is derived from the base class represented by **base\_class\_name**. The **access\_specifier** specifies the visibility, which indicates how the data members and member functions of a base class can be accessed by the objects of the derived class. An access specifier can be **public** or **private**. If no access specifier is defined, it is **private** by default.

For example:

```
class derived_demo : public base_demo
{
    data members of derived_demo;
    public:
        member functions of derived_demo;
}
```

In the above example, the base class **base\_demo** is publicly inherited by the derived class **derived\_demo**. The public members of the base class can be accessed by the objects of the derived class because they become public to the derived class.

A base class can be privately inherited, as shown in the following example:

```
class derived_demo : private base_demo
{
    data members of derived_demo;
    public:
        member functions of derived_demo;
};
```

OR

```
class derived_demo : base_demo//Private by default
{
    data members of derived_demo;
    public:
        member functions of derived_demo;
};
```

You can use any of the above examples for inheriting a base class. The base class **base\_demo** is privately inherited by the derived class **derived\_demo**. The public members of the class **base\_demo** become private to the class **derived\_demo**. They can only be accessed by member functions of the derived class but they cannot be directly accessed by its objects.

In C++, inheritance is of the following five types:

1. Single inheritance
2. Multilevel inheritance
3. Multiple inheritance
4. Hierarchical inheritance
5. Hybrid inheritance

## Single Inheritance

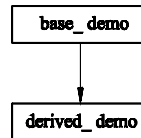
When a class is inherited from a single base class, it is called single inheritance.

For example:

```
class base_demo
{
    int a;//Private member
    public:
        int b;
        void get(int x, int y)
        {
            a=x;
            b=y;
        }
        void show( );
};
class derived_demo : public base_demo
{
    -----;
    -----;
    -----;
};
```

In the above example, the class **derived\_demo** publicly inherits the properties from a single

**Evaluation Chapter. Do not copy. Visit [www.cadclm.com](http://www.cadclm.com) for details**



### Example 1

The following program will calculate the area of a rectangle and display the resultant value on the screen. The line numbers on the right are not a part of the program and are for reference only.

```

//Write a program to find the area of a rectangle
#include<iostream>
using namespace std;
class rectangle
{
    int length;
public:
    int width;
    void getdata( )
    {
        length=10;
        width=10;
    }
    int getlength( )
    {
        return length;
    }
    void show( )
    {
        cout<<"Length = "<<length<<endl;
        cout<<"Width = "<<width<<endl;
    }
};
class rect_area : public rectangle
{

```

```

        int result;                                26
    public:                                        27
        void area( )                               28
        {                                          29
            result= width * getlength( );          30
        }                                          31
        void display( )                           32
        {                                          33
            cout<<"Length = "<<getlength( )<<endl; 34
            cout<<"Width  = "<<width<<endl;         35
            cout<<"Area   = "<<result<<endl;         36
        }                                          37
};                                                38
int main( )                                       39
{                                                40
    rect_area r1;                                41
    r1.getdata( );                               42
    r1.area( );                                   43
    r1.getlength( );                             44
    r1.display( );                               45
    r1.width=20;                                  46
    r1.area( );                                   47
    r1.display( );                               48
    return 0;                                     49
}                                                  50

```

### Explanation

Line 6

#### **int length;**

In this line, **length** is declared as an integer type variable. The variable **length** is the **private** data member of the class **rectangle**. It can be accessed only with the help of the **public** member functions of the same class.

Lines 14 to 17

#### **int getlength( )**

```

{
    return length;
}

```

These lines contain the definition of the member function **getlength( )**. Whenever a call is made to this function, it will return a value of the variable **length**.

Line 24

#### **class rect\_area : public rectangle**

In this line, the class **rect\_area** publicly inherits the properties (data member, member functions) of the base class **rectangle**. Only the **public** data members and member functions of the class **rectangle** (base class) are inherited by the objects of the class **rect\_area**.

(derived class). The **private** data members can also be accessed but only with the **public** member functions of the class **rectangle**.

Line 26

**int result;**

The **result** is declared as an integer type variable. This data member is declared in the **private** section of the class **rect\_area**. It can be accessed directly by the objects of the same class.

Lines 28 to 31

**void area()**

{

**result= width \* getlength();**

}

These lines contain the definition of the member function **area()**. In the body of the function, the **public** data member **width** of the base class **rectangle** is accessed directly but the **private** data member **length** of the base class is accessed with the help of a **public** member function **getlength()**. After multiplication, the resultant value is assigned to the variable **result**.

Line 41

**rect\_area r1;**

Here, variable **r1** is declared as an object of the derived class **rect\_area**.

Line 46

**r1.width=20;**

In this line, value 20 is assigned to the **public** data member **width** of the class **rectangle**, which is directly accessed by an object **r1** of the derived class **rect\_area**.

The output of the program is as follows:

Length = 10

Width = 10

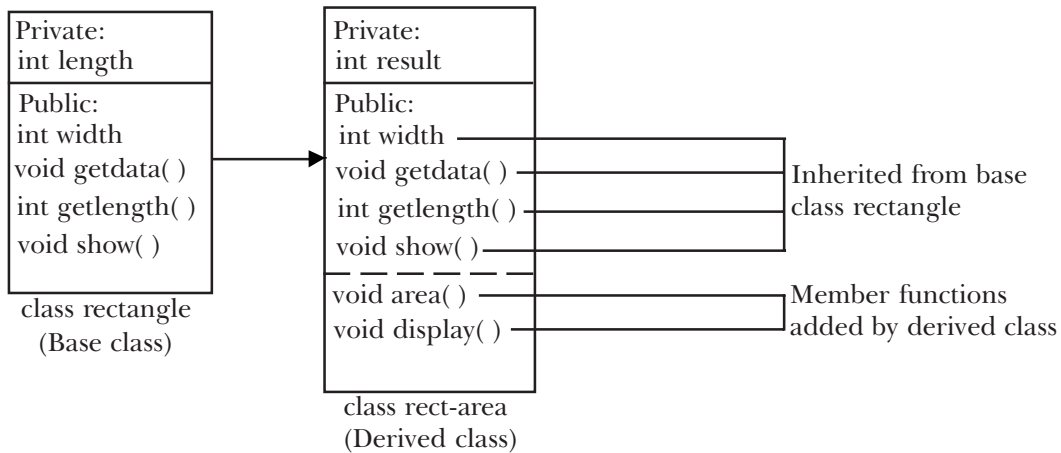
Area = 100

Length = 10

Width = 20

Area = 200

Figure 10-2 represents single inheritance (public), as discussed in Example 1.



**Figure 10-2** Representation of Example 1

In the previous program, the base class **rectangle** was publicly inherited by the derived class **rect\_area**. In this section, you will learn about **single inheritance** with **private** derivation. When a class is privately inherited, the **public** data members and member functions of the base class become **private** in the derived class. So, these members are not directly accessed by the objects of the derived class.

The following example illustrates the use of single inheritance (private).

## Example 2

Write a program to calculate the area of a rectangle.

The following program will calculate the area of a rectangle and display the resultant value on the screen.

```

//Write a program to calculate the area of a rectangle
#include<iostream>
using namespace std;
class rectangle
{
    int length;
    public:
        int width;
        void getdata()
        {
            cout<<"Enter the length of a rectangle: ";
            cin>>length;
            cout<<"Enter the width of a rectangle: ";
            cin>>width;
        }
}
  
```

```

        int getlength( )           16
        {                           17
            return length;          18
        }                           19
        void show( )               20
        {                           21
            cout<<"Length = "<<length<<endl; 22
            cout<<"Width  = "<<width<<endl;  23
        }                           24
    };                               25
    class rect_area : private rectangle//Private derivation 26
    {                               27
        int result;                 28
        public:                     29
            void area( )             30
            {                         31
                getdata( );          32
                result= width * getlength( ); 33
            }                         34
            void display( )          35
            {                         36
                show( );              37
                cout<<"Result = "<<result<<endl; 38
            }                         39
    };                               40
    int main( )                     41
    {                               42
        rect_area r1;               43
        r1.area( );                  44
        r1.display( );               45
        return 0;                    46
    }                               47

```

### Explanation

Line 26

#### **class rect\_area : private rectangle**

In this line, the base class **rectangle** is privately inherited by the derived class **rect\_area**. Now, the **public** members such as **width**, **getlength( )**, and so on of the class **rectangle** become **private** members of the derived class **rect\_area**. So, the objects of the class **rect\_area** do not access these members directly.

The output of the program is as follows:

Enter the length of a rectangle: 10

Enter the width of a rectangle: 20

Length =10

Width = 20

Area = 200

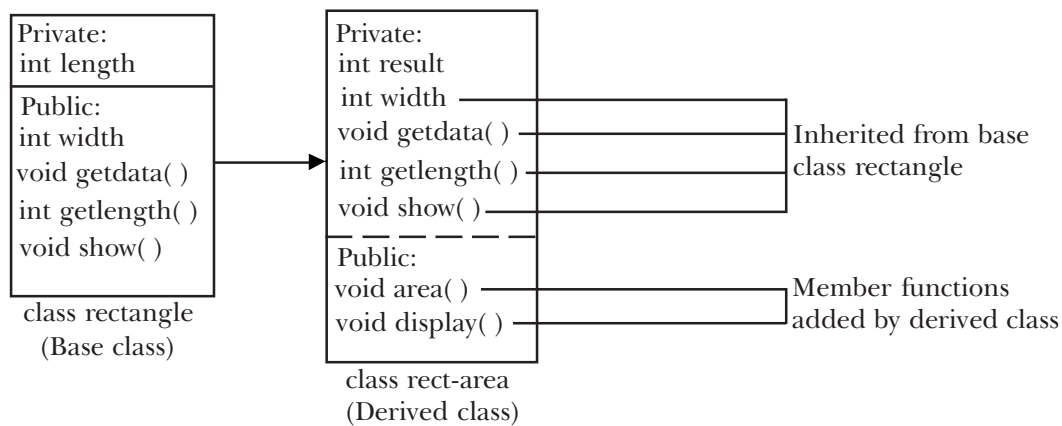


The output will be displayed on the screen as follows:

```

c:\C++\C++_programs\Ch_10\ch10_pr...
Enter the length of a rectangle: 10
Enter the width of a rectangle: 20
Length = 10
Width = 20
Result = 200
Press any key to continue.
  
```

Figure 10-3 represents single inheritance (private), as discussed in Example 2.



*Figure 10-3 Representation of Example 2*

### Making the Private Members of a Base Class Inheritable

In the earlier sections, you have seen that the **private** members of a base class cannot be inherited and are not directly available to the derived class. In case the **private** data needs to be inherited by a derived class, you can simply do it by modifying the visibility limit of a **private** member by making it **public** or **protected**.

When the visibility limit of **private** members of a base class is changed to **public**, the members are accessible by all the functions within the same class and also, by the functions of the other classes. This method makes the members **public**, and this eliminates the feature of data hiding.

Another way is to change the visibility limit of the **private** members of a base class by declaring them as **protected** members. This results in only a limited access to the members of a class. The **protected** members of a class are only accessed by the members of the same class and any class that is immediately derived from it. The syntax for defining **protected** members in the definition of a class is given next.

```

class class_name
{
    protected:
        data members;
        member functions( );
};

```

In the above syntax, the **protected** members can be accessed by the members of the class specified by **class\_name** and by the members of the class that is immediately derived from the class **class\_name**.

When the **protected** members are inherited in the **public** mode, they become protected in the derived class. So, they are directly accessed by the member functions of the derived class and can be further inherited.

For example:

```

class rectangle
{
    protected:
        int length;
    public:
        int width;
        void getdata( );
        void show( );
};
class rect_area : public rectangle
{
    int result;
    public:
    void area( )
    {
        getdata( );
        result=width * length;
    }
    void display( );
};

```

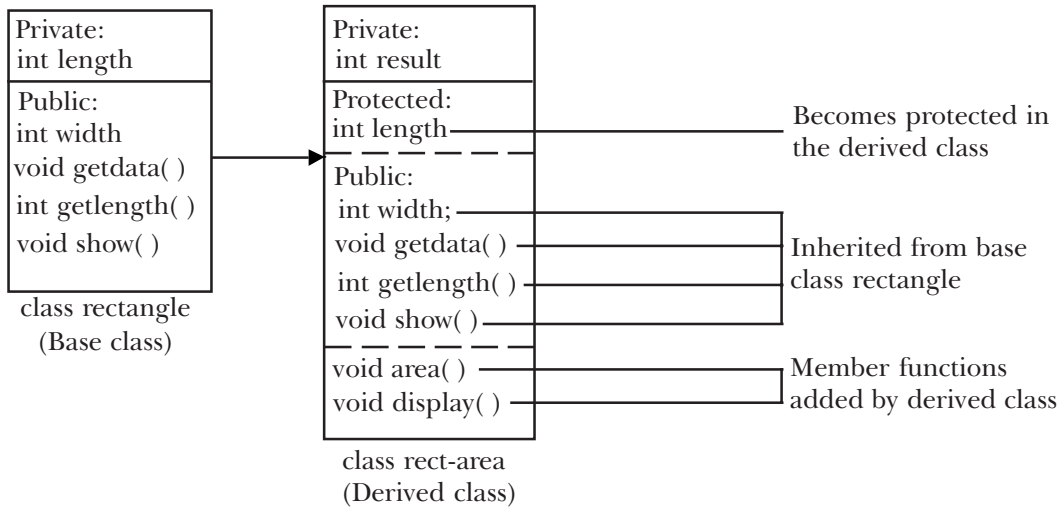
In the above example, the class **rectangle** is publicly inherited by the class **rect\_area**. The **protected** member **length** in the base class **rectangle** becomes **protected** in the derived class **rect\_area**, as shown in Figure 10-4. So it can be directly accessed by the member functions of the derived class, as shown in the following statement:

```

result=width * length;

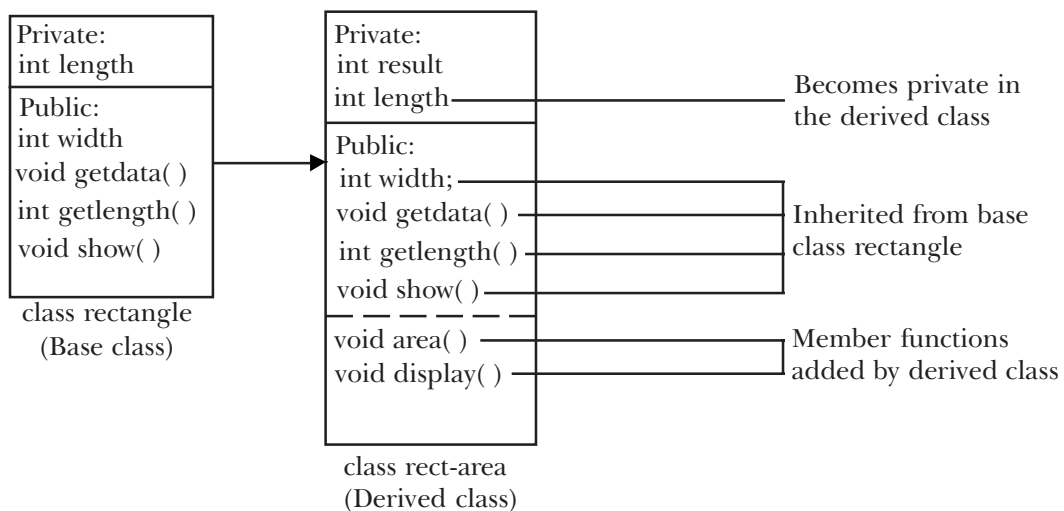
```

In this statement, the **protected** data member **length** is directly accessed by the member function **area()** in the derived class.



**Figure 10-4** Protected members inherited in the **public** mode

When the **protected** members are inherited in the **private** mode, they become **private** in the derived class. They are accessed by the member functions of the derived class but they cannot be further inherited. Figure 10-5 shows a representation of a protected member inherited in the **private** mode.



**Figure 10-5** Protected members inherited in the **private** mode

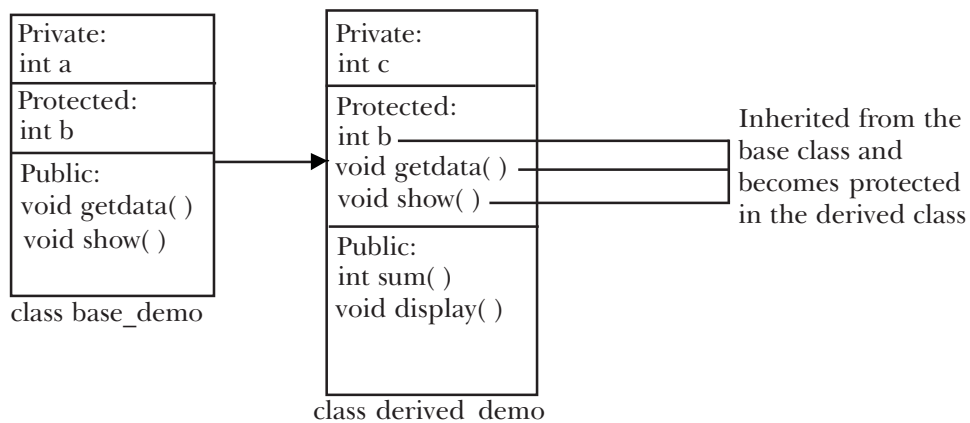
As shown in Figure 10-5, the **protected** data member **length** of the base class became **private** in the derived class.

A base class can also be inherited in the **protected** mode. The **public** and **protected** members of the base class become **protected** in the derived class. So, they are directly accessed by the member functions of the derived class and are also available for further inheritance.

For example:

```
class base_demo
{
    int a;    //By default private
protected:
    int b;
public:
    void getdata( );
    void show( );
};
class derived_demo : protected base_demo
{
    int c;
public:
    int sum( );
    void display( );
};
```

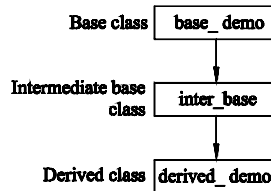
In the above example, the base class **base\_demo** is inherited by the class **derived\_demo** in the **protected** mode. The **protected** data member **b** and the **public** member functions, **getdata( )** and **show( )** become **protected** members of the class **derived\_demo**, as shown in Figure 10-6.



**Figure 10-6** A base class inherited in the **protected** mode

## Multilevel Inheritance

When a class is derived from another derived class, it is known as multilevel inheritance, as shown in Figure 10-7.



*Figure 10-7 Multilevel inheritance*

In the above figure, the class **base\_demo** represents the base class, which is inherited by another class **inter\_base**. Now, the derived class **inter\_base** is inherited by another class **derived\_demo**. Here, the **inter\_base** class works as a base class for the class **derived\_demo**. The class **inter\_base** is also known as the **intermediate base class**.

The syntax for declaring classes that contain multilevel inheritance is as follows:

```

class base_demo
{
    data members;
    public:
        member functions;
};
class inter_demo : public base_demo
{
    data members;
    public:
        member functions;
};
class derived_demo : public inter_demo
{
    data members;
    public:
        member functions;
};
  
```

In the above syntax, the class **base\_demo** represents the base class, which is publicly inherited by another class **inter\_demo**. Now, the class **inter\_demo** is publicly inherited by another class **derived\_demo**. Here, the class **inter\_demo** serve as a base class for the class **derived\_demo**.

The following example illustrates the use of multilevel inheritance.

### Example 3

Write a program to calculate the average of points by using multilevel inheritance.

The following program will prompt the users to enter their information and also the points scored in three subjects. The program will then calculate the average and display the information as well as the average on the screen.

```
//Write a program to calculate the average of the points entered by the user      1
#include<iostream>                                                                2
using namespace std;                                                            3
class student//Base class                                                       4
{                                                                                5
    char name[30];                                                              6
    int stu_id;                                                                7
    char course[10];                                                           8
    public:                                                                    9
        void getdata( );                                                       10
        void showdata( );                                                      11
};                                                                              12
void student :: getdata( )                                                     13
{                                                                              14
    cout<<"Enter name: ";                                                     15
    cin>>name;                                                                16
    cout<<"Enter the student ID: ";                                           17
    cin>>stu_id;                                                              18
    cout<<"Enter course: ";                                                  19
    cin>>course;                                                             20
}                                                                              21
void student :: showdata( )                                                    22
{                                                                              23
    cout<<"Name: "<<name<<endl;                                              24
    cout<<"Student ID: "<<stu_id<<endl;                                       25
    cout<<"Course: "<<course<<endl;                                         26
}                                                                              27
class points : public student//First level of inheritance                     28
{                                                                              29
    protected:                                                                30
        float sub[3];                                                         31
    public:                                                                    32
        void get_points( );                                                   33
        void show_points( );                                                 34
};                                                                              35
```

```

void points :: get_points( )           36
{                                     37
    for(int i=0; i<3; i++)           38
    {                                 39
        cout<<"Enter the points scored in subject"<<i+1<<" : "; 40
        cin>>sub[i];                 41
    }                                 42
}                                     43
void points :: show_points( )         44
{                                     45
    for(int j=0; j<3; j++)           46
    {                                 47
        cout<<"Points scored in subject"<<j+1<<" are: "; 48
        cout<<sub[j]<<endl;          49
    }                                 50
}                                     51
class average : public points//Second level of inheritance 52
{                                     53
    float avrg;                       54
    public:                           55
        void result( );               56
};                                     57
void average :: result( )             58
{                                     59
    avrg= (sub[0]+sub[1]+sub[2])/3;    60
    showdata( );                      61
    show_points( );                   62
    cout<<"Average: " <<avrg<<endl;  63
}                                     64
int main( )                           65
{                                     66
    average a1;                       67
    a1.getdata( );                    68
    a1.get_points( );                 69
    cout<<"-----" <<endl;         70
    a1.result( );                     71
    return 0;                         72
}                                     73

```

### Explanation

Lines 4 to 12

**class student**

```

{
    char name[30];
    int stu_id;
    char course[10];

```

```

    public:
        void getdata( );
        void showdata( );
};

```

These lines contain the definition of the class **student**, which contains three data members **name**, **stu\_id**, **course** and two member functions, **getdata()** and **showdata()**. Here, the class **student** is treated as a base class.

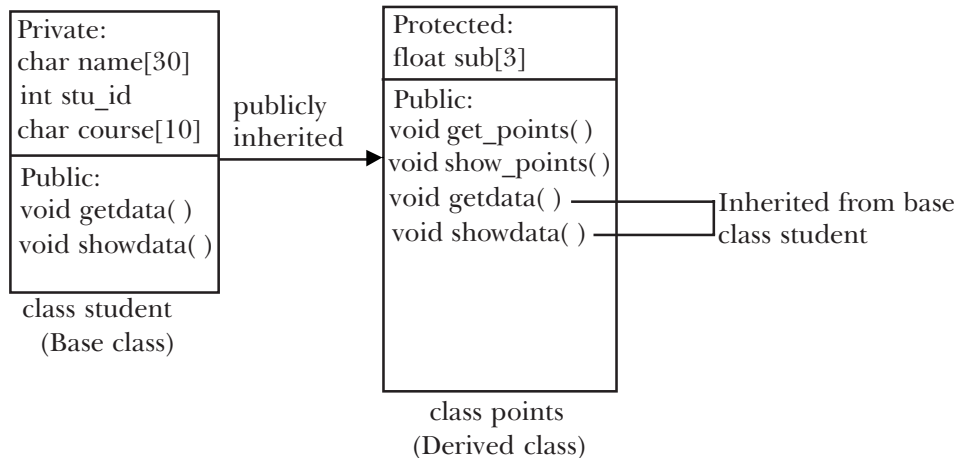
Lines 28 to 35

```

class points : public student
{
    protected:
        float sub[3];
    public:
        void get_points( );
        void show_points( );
};

```

The class **points** publicly inherits the properties of the base class **student**. So, the private data members of the base class **student** are not directly accessed by the objects of the derived class **points**. The definition of the class **points** contains a **protected** data member **sub**, which is an array of a **float** type with the maximum size 3. It also contains two member functions, **get\_points()** and **show\_points()**. After inheritance, the class **points** contains data members and member functions, as shown in Figure 10-8.



**Figure 10-8** First level of inheritance

The above figure represents the first level of inheritance in which the base class **student** is inherited by the class **points**. After **public** inheritance, the **public** members of the base class become the **public** members of the derived class.



Lines 52 to 57

```
class average : public points
{
    float avrg;
    public:
        void result( );
};
```

The derived class **points** is treated as a base class for the class **average**. The class **average** publicly inherits the properties of the class **points**. So, the **protected** members of the class **points** can be directly accessed by the objects of the derived class **average**. After inheritance, the class **average** contains data members and member functions, as shown in Figure 10-9.

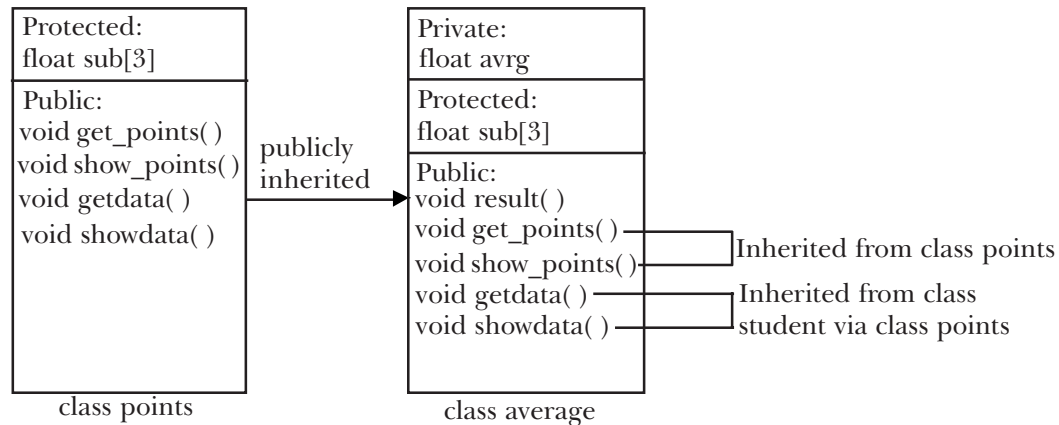


Figure 10-9 Second level of inheritance

The above figure represents the second level of inheritance in which the derived class **points** is inherited by another class **average**. Here, the class **points**, which is a derived class, is treated as a base class for the class **average**.

Line 67

```
average a1;
```

In this line, the variable **a1** is declared as an object of the class **average**.

Line 68

```
a1.getdata( );
```

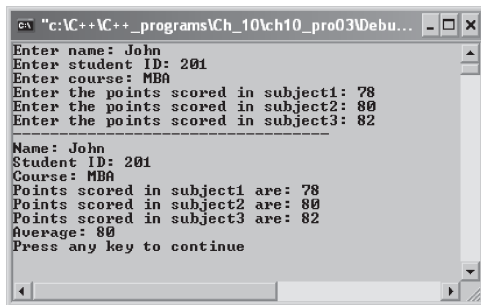
In the first level of inheritance, all the **public** members of the base class **student** become **public** members of the derived class **points**. In the second level, all the **public** members of the class **points** become **public** members of the class **average**. So now, the **getdata( )** member function is directly accessed by an object **a1** of the class **average**.

The output of the program is as follows:

```
Enter name: John
Enter the student ID: 201
Enter course: MBA
Enter the points scored in subject1: 78
Enter the points scored in subject2: 80
Enter the points scored in subject3: 82
-----
```

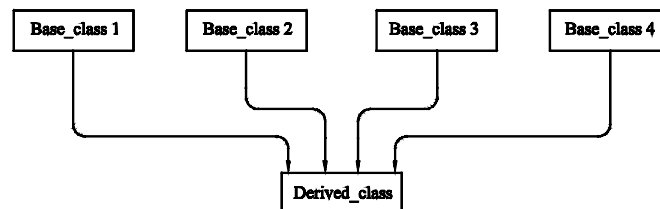
```
Name: John
Student ID: 201
Course: MBA
Points scored in subject1 are: 78
Points scored in subject2 are: 80
Points scored in subject3 are: 82
Average: 80
```

The output will be displayed on the screen as follows:



## Multiple Inheritance

When a class inherits the properties from two or more base classes, it is known as multiple inheritance. In multiple inheritance, a class is derived by inheriting the properties of two or more than two base classes, as shown in Figure 10-10.



*Figure 10-10 Representation of multiple inheritance*

This figure represents multiple inheritance. In this figure, the derived class, which is specified by **Derived\_class**, inherits the properties from the base classes specified by **Base\_class1** to **Base\_classN**.

The syntax is as follows:

```
class Derived_class : access_specifier Base_class1, ....., access_specifier Base_classN
{
    data_members;
    public:
        member functions;
};
```

In this syntax, the base classes that are specified by **Base\_class1** to **Base\_classN** are separated by commas.

For example:

```
class average : public student, public points
{
    float avrg;
    public:
        void result( );
};
```

In the above example, the class **average** publicly inherits the properties of two classes **student** and **points**.

The following example illustrates the use of multiple inheritance.

#### Example 4

Write a program to calculate the average of points by using multiple inheritance.

The following program will prompt the users to enter their information and also the points scored in three subjects. The program will then calculate the average and display the information as well as the average on the screen.

```
//Write a program to calculate the average of the points entered by the user    1
#include<iostream>                                                                2
using namespace std;                                                            3
class student                                                                    4
{                                                                                  5
    char name[30];                                                                6
    int stu_id;                                                                    7
    char course[10];                                                              8
```

```

        public:
            void getdata( );
            void showdata( );
};
void student :: getdata( )
{
    cout<<"Enter name: ";
    cin>>name;
    cout<<"Enter the student ID: ";
    cin>>stu_id;
    cout<<"Enter course: ";
    cin>>course;
}
void student :: showdata( )
{
    cout<<"Name: "<<name<<endl;
    cout<<"Student ID: "<<stu_id<<endl;
    cout<<"Course: "<<course<<endl;
}
class points
{
    protected:
        float sub[3];
    public:
        void get_points( );
        void show_points( );
};
void marks :: get_points( )
{
    for(int i=0; i<3;i++)
    {
        cout<<"Enter the points scored in subject"<<i+1<<" ";
        cin>>sub[i];
    }
}
void marks :: show_points( )
{
    for(int j=0; j<3; j++)
    {
        cout<<"Points scored in subject"<<j+1<<" are: ";
        cout<<sub[j]<<endl;
    }
}
class average : public student, public points
{
    float avrg;
    public:

```

```

        void result( );           56
    };                             57
void average :: result( )        58
{                                  59
    avrg = (sub[0]+sub[1]+sub[2])/3; 60
    showdata( );                  61
    show_points( );               62
    cout<<"Average: "<<avrg<<endl; 63
}                                  64
int main( )                      65
{                                  66
    average a1;                   67
    a1.getdata( );                68
    a1.get_points( );             69
    cout<<"-----"<<endl;        70
    a1.result( );                 71
    return 0;                     72
}                                  73

```

### Explanation

Line 52

**class average : public student, public points**

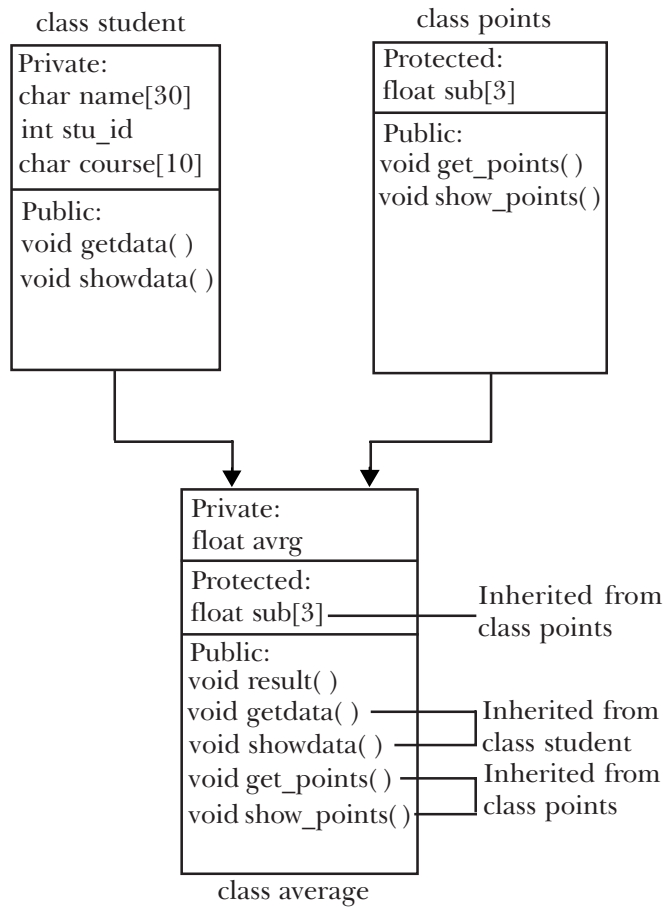
Here, the class **average** publicly inherits the properties of the class **student** and the class **points**. After inheritance, the class **average** contains data members and member functions, as shown in Figure 10-11.

The output of the program is as follows:

```

Enter name: John
Enter student ID: 201
Enter course: MBA
Enter the points scored in subject1: 78
Enter the points scored in subject2: 80
Enter the points scored in subject3: 82
-----
Name: John
Student ID: 201
Course: MBA
Points scored in subject1 are: 78
Points scored in subject2 are: 80
Points scored in subject3 are: 82
Average: 80

```



**Figure 10-11** Members of the class average after inheritance

In multiple inheritance, when a class inherits the properties from more than one base class, the problem of ambiguity may occur. When a function with the same syntax (same name, same return type) exists in more than one base class, it is known as ambiguity.

For example:

```

class A
{
    public:
        void display( )
        {
            cout<<"You are in class A"<<endl;
        }
};
    
```

```
class B
{
    public:
        void display( )
        {
            cout<<"You are in class B"<<endl;
        }
};
```

In the above code, both the classes **A** and **B** contain a member function **void display()** whose syntax is the same in both the classes.

```
class C : public A, public B
{
    public:
        void show( )
        {
            display( );
        }
};
```

The class **C** publicly inherits the properties from the base classes **A** and **B**. When a call is made to the function **display()** in the **public** member function **show()** in class **C**, it will give an error. This error is because of an ambiguity in the program. This ambiguity can be removed by using the name of a class and a scope resolution operator (**::**) with a name of the function, as follows:

```
class C : public A, public B
{
    public:
        void show( )
        {
            A :: display( );
        }
};
```

In the above code, the class name **A** and the scope resolution operator (**::**) is used with the function **display()**. This means that the member function **display()** of the class **A** will be called whenever a call is made to the function **show()** of the class **C**.

In the above case, you have seen that the problem of ambiguity arises when a single class inherits the properties from more than one base class. This problem can also arise in the case of single inheritance in which a single class inherits the properties from a single base class.

For example:

```
class X
{
    public:
        void show( )
        {
            cout<<"You are in class A"<<endl;
        }
};
class Y : public X
{
    public:
        void show( )
        {
            cout<<"You are in class B"<<endl;
        }
};
```

In this example, the member function **show( )** of the derived class **Y** overrides the member function **show( )** of the base class **X**. So, whenever a call is made to the **show( )** function with an object of class **Y**, the compiler will invoke the **show( )** function defined in the derived class.

You can also invoke the function defined in the base class by specifying the class name and a scope resolution operator with the function name, as follows:

```
int main( )
{
    Y y1;
    y1.show( );//make a call to show( ) in derived class Y
    y1.X::show( );//make a call to show( ) in base class X
    y1.Y::show( );//make a call to show( ) in derived class Y
    return 0;
}
```

## Hierarchical Inheritance

In hierarchical inheritance, the classes are arranged in a hierarchy based on the properties they contain. The base class is at the top of the hierarchy and contains all the properties that are common to the subclasses. A subclass can be derived by inheriting the properties of the base class. In hierarchical inheritance, a subclass can function as a base class for the lower level of classes. Figure 10-12 represents a hierarchical classification of students in a college. In this figure, the **Students** class contains the properties that are common to the subclasses (Management, Computers, and Engineering). The **Management** class serves as a base class for the lower level of classes (Marketing, Finance, and Human Resource).



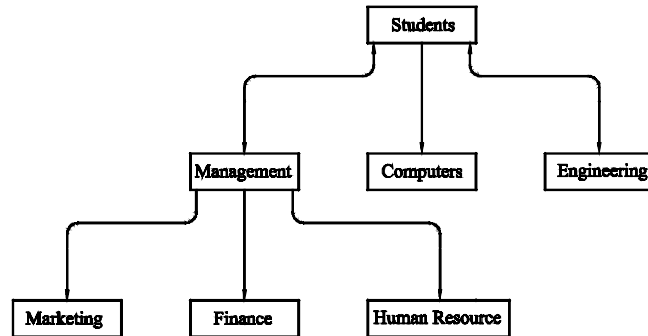


Figure 10-12 Representation of hierarchical inheritance

## Hybrid Inheritance

In general terms, a variety, which is a composite of mixed origins is called a hybrid form. In the same way, in C++, when two or more than two types of inheritances are used to design a program, it is known as hybrid inheritance. For example, if in a program two types of inheritance, multilevel and multiple are used, this mixed type of inheritance is known as hybrid inheritance, see Figure 10-13.

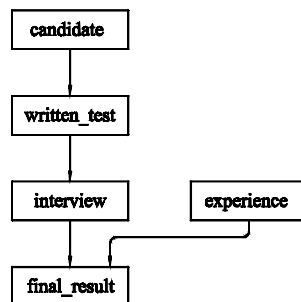


Figure 10-13 Example of hybrid inheritance

The following example illustrates the use of hybrid inheritance.

### Example 5

Write a program to calculate the final result of candidates in an interview by using hybrid inheritance.

The following program will prompt the user to enter his information and the points scored in the tests, calculate the final result, and display it on the screen.

```

//Write a program to illustrate the working of hybrid inheritance      1
#include<iostream>                                                    2
using namespace std;                                                  3
class candidate                                                        4
{                                                                      5
    protected:                                                        6
    char name[20];                                                    7
    int ref_num;                                                       8
    public:                                                            9
        void getdata( )                                              10
        {                                                            11
            cout<<"Enter the name of the candidate: ";              12
            cin>>name;                                                13
            cout<<"Enter the reference number: ";                    14
            cin>>ref_num;                                             15
        }                                                            16
        void showdata( )                                             17
        {                                                            18
            cout<<"Name: "<<name<<endl;                               19
            cout<<"Reference number: "<<ref_num<<endl;                20
        }                                                            21
};                                                                      22
class written_test : public candidate//Level 1 of multilevel inheritance 23
{                                                                      24
    protected:                                                        25
    float apt_points;                                                 26
    float tech_points;                                                27
    public:                                                            28
        void getpoints( )                                           29
        {                                                            30
            cout<<"Enter the points scored in the Aptitude test: "; 31
            cin>>apt_points;                                          32
            cout<<"Enter the points scored in the Technical test: "; 33
            cin>>tech_points;                                         34
        }                                                            35
        void showpoints( )                                           36
        {                                                            37
            cout<<"Points scored in the Aptitude test are: "        38
                <<apt_points<<endl;
            cout<<"Points scored in the Technical test are: "        39
                <<tech_points;
            cout<<endl;                                              40
        }                                                            41
};                                                                      42
class interview: public written_test//Level 2                        43
{                                                                      44
    protected:                                                        45

```

```

float tech_score; 46
float hr_score; 47
public: 48
    void getscore( ) 49
    { 50
        cout<<"Enter the points scored "
        "in the Technical interview: "; 51
        cin>>tech_score; 52
        cout<<"Enter the points scored in the HR interview: "; 53
        cin>>hr_score; 54
    } 55
    void showscore( ) 56
    { 57
        cout<<"Points scored in the Technical interview are: "
        <<tech_score<<endl; 58
        cout<<"Points scored in the HR interview are: "
        <<hr_score<<endl; 59
    } 60
}; 61
class experience 62
{ 63
    protected: 64
    float years; 65
    public: 66
        void get_exp( ) 67
        { 68
            cout<<"Enter experience: "; 69
            cin>>years; 70
        } 71
        void show_exp( ) 72
        { 73
            cout<<"Experience in years: "<<years<<endl; 74
        } 75
}; 76
class final_result: public interview, public experience//Multiple inheritance 77
{ 78
    float result; 79
    public: 80
        void show_result( ) 81
}; 82
void final_result:: show_result( ) 83
{ 84
    result= apt_points + tech_points + tech_score + hr_score + years; 85
    showdata( ); 86
    showpoints( ); 87
    showscore( ); 88
    show_exp( ); 89
}

```

```

        cout<<"Total score: "<<result<<endl;
    }
int main( )
{
    final_result candidate_1;
    candidate_1.getdata( );
    candidate_1.getpoints( );
    candidate_1.getscore( );
    candidate_1.get_exp( );
    candidate_1.show_result( );
    return 0;
}

```

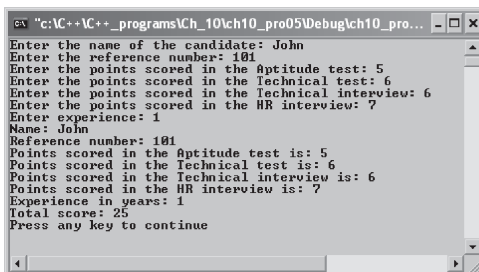
The output of the program is as follows:

```

Enter the name of the candidate: John
Enter the reference number: 101
Enter the points scored in the Aptitude test: 5
Enter the points scored in the Technical test: 6
Enter the points scored in the Technical interview: 6
Enter the points scored in the HR interview: 7
Enter experience: 1
Name: John
Reference number: 101
Points scored in the Aptitude test are: 5
Points scored in the Technical test are: 6
Points scored in the Technical interview are: 6
Points scored in the HR interview are: 7
Experience in years: 1
Total score: 25

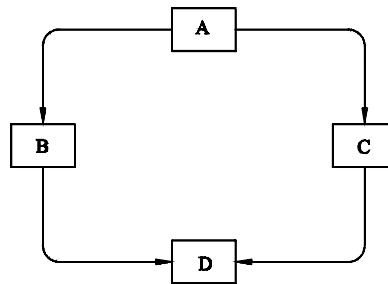
```

The output will be displayed on the screen as follows:



## VIRTUAL BASE CLASSES

In the previous section, you learned about hybrid inheritance, in which two or more than two types of inheritances were used to design a program. But sometimes, when all the three types of inheritances multiple, multilevel, and hierarchical are used in hybrid inheritance to design a program, a problem of ambiguity may occur, see Figure 10-14.



*Figure 10-14 Multipath inheritance*

In the above figure, classes **B** and **C** both inherit the base class **A**, whereas the class **D** inherits the classes **B** and **C**. Here, the class **D** indirectly inherits the properties of the base class **A** twice through classes **B** and **C**. So, all the **public** and the **protected** members of the base class **A** are inherited twice by the class **D**. Therefore, the class **D** contains duplicate copies of the members of the base class **A**. This problem of duplicacy or ambiguity can be solved by declaring the base class **A** as a **virtual** class.

When a common base class is declared as a **virtual** base class, only one copy of that class is inherited by the derived class through all the multiple paths. The syntax for declaring a **virtual** base class is as follows:

```

class A
{
    class definition;
};
class B : public virtual A
{
    class definition;
};
class C : virtual public A
{
    class definition;
};
class D : public B, public C
{
    class definition;
};
  
```

In the above syntax, the base class **A** is declared as a **virtual** base class, which is publicly inherited by the classes **B** and **C**. Here, class **B** and **C** are publicly inherited by the derived class **D**. The class **D** indirectly inherits the base class **A** through the two paths, but class **D** contains only one copy of the members of the base class **A**. This is because the base class **A** has been declared as a **virtual** base class.

In the previous syntax, the following statements

```
public virtual A
or
virtual public A
```

are the same. You can use the keyword **virtual** before or after the access specifier.

The following example illustrates the use of the **virtual** base class.

### Example 6

Write a program to illustrate the working of the **virtual** base class.

The following program will multiply three numbers and display the resultant value on the screen.

```
//Write a program to multiply three numbers          1
#include<iostream>                                     2
using namespace std;                                  3
class A                                               4
{                                                     5
    public:                                           6
        int a;                                       7
};                                                    8
class B : public virtual A                            9
{                                                    10
    public:                                           11
        int b;                                       12
};                                                    13
class C : virtual public A                           14
{                                                    15
    public:                                           16
        int c;                                       17
};                                                    18
class D : public B, public C                          19
{                                                    20
    public:                                           21
        int mul;                                    22
};                                                    23
int main( )                                           24
{                                                    25
    D d1;                                             26
    d1.a=10;                                          27
    d1.b=10;                                          28
    d1.c=10;                                          29
    d1.mul= d1.a * d1.b * d1.c;                     30
```

```
        cout<<"The result of multiplication is: "<<d1.mul<<endl;    31
        return 0;                                                  32
    }                                                                33
```

The output of the program is as follows:  
The result of multiplication is: 1000

### Self-Evaluation Test

Answer the following questions and then compare them to the answers given at the end of this chapter:

1. The technique of creating a new class from an existing class is known as \_\_\_\_\_.
2. A \_\_\_\_\_ class can inherit some or all the properties of a \_\_\_\_\_ class.
3. When a class is privately inherited, the **public** members of the base class become \_\_\_\_\_ in the derived class.
4. When a class is derived from another derived class, it is known as \_\_\_\_\_ inheritance.
5. The problem that occurs during multiple inheritance is known as \_\_\_\_\_.

### Review Questions

Answer the following questions:

1. Define inheritance. What is the main advantage of inheritance?
2. Explain the different types of inheritance.
3. Differentiate between **public** and **private** members of a class. Explain the difference with a suitable example.
4. Explain multiple inheritance with a suitable example. What is the main problem that occurs during multiple inheritance?
5. What do you mean by a **virtual** base class?

**Exercise****Exercise 1**

Write a program to calculate the final result of some students. This program should contain the four classes, which are discussed next.

1. Class **student**, which contains two **protected** data members, **student\_id**, and **name**. It should also contain **public** member functions to accept the values of the data members from the user and display them on the screen. The class **student** should be declared as the **virtual** base class.
2. Class **test**, which publicly inherits the properties of the base class **student**. This class should contain a data member, **test\_points** and also two member functions to read the points scored from the user and display them on the screen.
3. Class **exam**, which publicly inherits the properties of the base class **student**. This class should contain a data member, **final\_points** and also two member functions to read the points scored from the user and display them on the screen.
4. Class **result**, which publicly inherits both the classes, **test** and **exam**. This class should contain a data member, **total\_result** and also two member functions to calculate the final result of the student and display it on the screen.

**Answers to Self-Evaluation Test**

1. inheritance, 2. derived, base, 3. **private**, 4. multilevel, 5. ambiguity