Chapter 12

Inheritance

12.1 Introduction

Reusability is an important feature of C++. The existing classes are used to create new classes, this mechanism is called inheritance. Using this feature the programmer can save time, money and effort. The existing class is called base class or parent class or super class and the new class is called derived class or child classes or subclass.

12.2 Defining derived classes

The syntax of derived class is:

```
class derived-class-name : visibility-mode base-class-name
{
        members of derived class.
};
```

The visibility-mode can be either private, protected or public. By default, the visibility-mode is private. The visibility-mode specifies whether the features of the base class are privately, protectedly or publically derived .

If the base class is privately inherited by the derived class , the public and protected members of the base class become private members of the derived class. The private members of the base class are never inherited .

If the base class is protectedly inherited by the derived class, the protected and public members of the base class become protected members of the derived class.

If the base class is publically inherited by the derived class, the protected members of the base class become protected members of the derived class and the public members of the base class become public members of the derived class.

12.3 Single Inheritance

In single inheritance, there is one base class and one derived class.

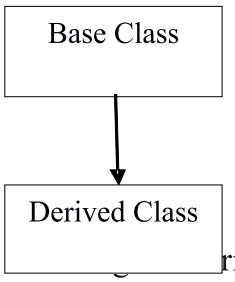


Fig. ritance

Program 12.1: Single inheritance

```
#include<iostream>
using namespace std;
class data
{
 protected:
    int x,y;
 public:
        void getdata(int a, int b)
        {
                x=a;
                y=b;
        }
        void showdata(void)
        {
                cout<<"x="<<x<<"\n";
                cout<<"y="<<y<<"\n";
        }
};
class maximum: public data
{
 public:
         void max(void)
         {
                if(x>y)
                cout<<"Maximum is:"<<x;
                else
                cout<<"Maximum is:"<<y;
         }
};

int main()
{
 maximum m;
 m.getdata(4,9);
 m.showdata();
 m.max();
return 0;
}
```

The output of the program 12.1 would be:

```
x=4
y=9
Maximum is: 9
```

In the above program, the base class 'data' has two protected data members x and y. These two data members can be accessed by the base class and its immediate derived class, but not outside to these two classes. The derived class 'maximum' compute maximum between these two data members. After public derivation of the base class, the derived class has the following members.

Derived class 'maximum'

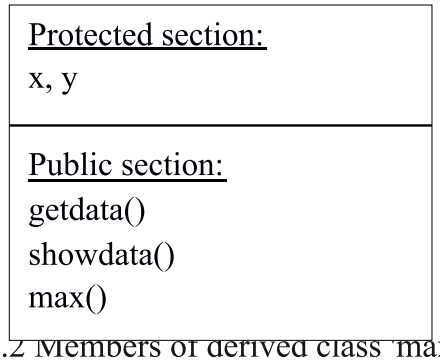


Fig. 12.2 Members of derived class 'maximum'

12.4 Multilevel inheritance

A class can be derived from another derived class.

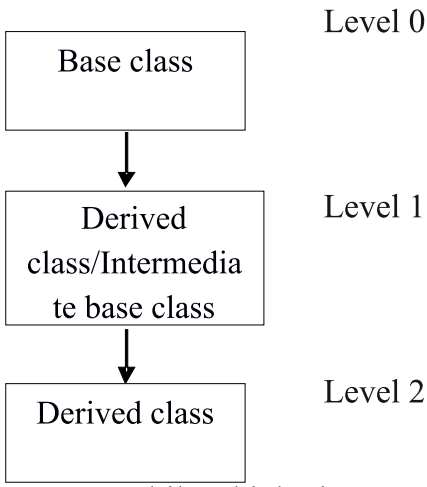


Fig. 12.3 Multilevel inheritance

There can be any number of levels in multilevel inheritance. The following program is an example of multilevel inheritance.

Program 12.2: Multilevel inheritance

```
#include<iostream>
using namespace std;
```

```
class data1
{
 protected:
    int x;
public:
            void get_x(int a)
            {
                    x=a;

            }
           void show_x(void)
           {
                    cout<<"x="<<x<<"\n";
            }
};

class data2:public data1
{
 protected:
    int y;
public:
            void get_y(int b)
            {
                    y=b;
             }
           void show_y(void)
           {
                   cout<<"y="<<y<<"\n";
           }
};
class addition: public data2
{
           int z;
public:
          void sum(void)
          {
                    z=x+y;
          }
          void show_z(void)
          {
                    cout<<"z="<<z<<"\n";
          }
```

```
};

int main()
{
        addition a;
        a.get_x(4);
        a.get_y(7);
        a.sum();
        a.show_x();
        a.show_y();
        a.show_z();
      return 0;
}
```

The output of the program 12.2 would be:

x=4
y=7
z=11

In the above program, the derived class 'data2' is derived from the base class 'data1' and this is the first level of derivation . The protected data member x of base class 'data1' become protected in derived class 'data2'. After first level of derivation, the derived class 'data2' has the following members.

Derived class 'data2'

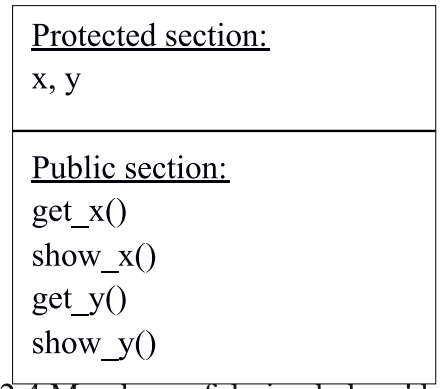


Fig. 12.4 Members of derived class 'data2'

The class 'addition' is derived from intermediate base class 'data2'. After this second level of derivation the derived class 'addition' has the following members.

Derived class 'addition'

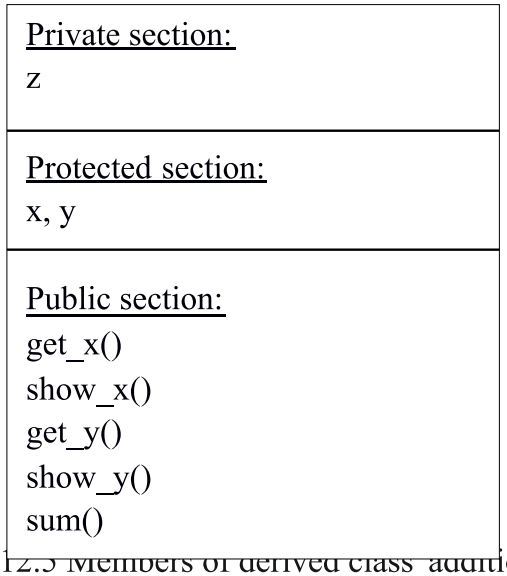


Fig. 12.5 Members of derived class 'addition'

12.5 Multiple Inheritance

When a class inherits the features of two or more classes, is known as multiple inheritance.

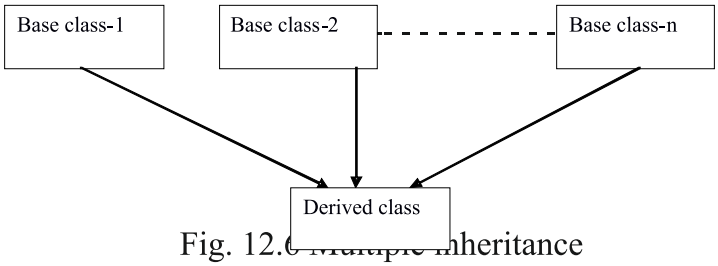


Fig. 12.6 Multiple inheritance

The syntax of derived class with multiple base classes is:
class derived_class : visibility Base_class-1, visibility Base_class-2, - - - - - - -
{
Members of derived class
};

The following program is an example of multiple inheritance.
Program 12.3: Multiple inheritance
#include<iostream>

```
using namespace std;
class B1
{
protected:
        int x;
public:
        void get_x(int a)
        {
                x=a;
        }
};
class B2
{
protected:
        int y;
public:
        void get_y(int b)
        {
                y=b;
        }
};
class D : public B1, public B2
{
        int z;
public:
        void multiply(void)
        {
                z=x*y;
        }
        void display(void)
        {
                cout<<“x=”<<x<<“\n”;
                cout<<“y=”<<y<<“\n”;
                cout<<“z=”<<z<<“\n”;
        }
};
int main()
{
        D d;
        d.get_x(5);
        d.get_y(3);
        d.multiply();
```

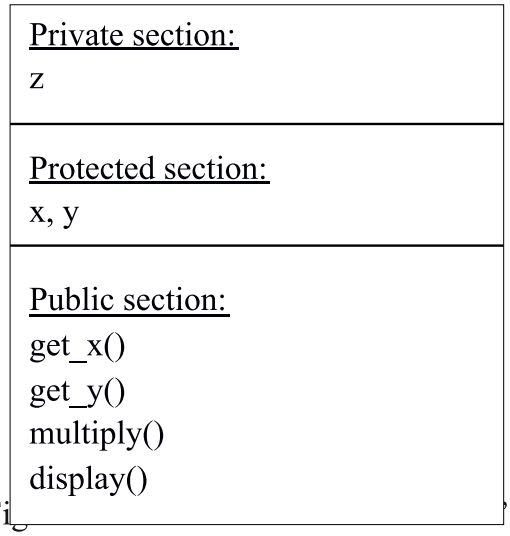
```
        d.display();
        return 0;
}
```

The output of the program 12.3 would be:

```
x=5
y=3
z=15
```

In the above program, the derived class 'D' inherits the members of base classes 'B1' and 'B2'. The derived class 'D' has the following members after this derivation.

Derived class 'D'



Fi

12.6 Hierarchical inheritance

When a base class is inherited by two or more derived classes, is known as hierarchical inheritance. As an example figure 12.8 shows hierarchical classification of persons in a school.

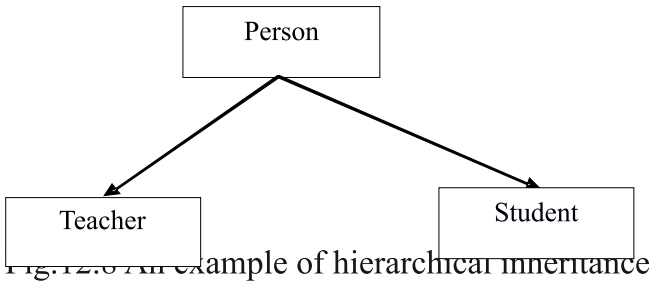


Fig.12.8 An example of hierarchical inheritance

Program 12.4: Hierarchical inheritance

```
#include<iostream>
#include<string.h>
using namespace std;
class person
{
 protected:
        char name[20];
        int age;
 public:
         void get_person(const char *n, int a)
         {
                strcpy(name,n);
                age=a;
         }
         void show_person(void)
         {
                cout<<"Name:"<<name<<"\n";
                cout<<"Age:"<<age<<"\n";
         }
};
class teacher : public person
{
 char post[10];
 public:
        void get_post(const char *p)
        {
                strcpy(post,p);
        }
        void show_teacher(void)
        {
                show_person();
                cout<<"post:"<<post<<"\n";
        }
};
class student : public person
{
         int standard;
```

```
public:
        void get_standard(int s)
        {
                standard=s;
         }
         void show_student(void)
         {
                show_person();
                cout<<"Standard:"<<standard<<"\n";
        }
};

int main()
{
          teacher t;
          t.get_person("Ram",30);
          t.get_post("TGT");
          student s;
          s.get_person("Shyam",17);
          s.get_standard(12);
          t.show_teacher();
          s.show_student();
        return 0;
}
```

The output of the program 12.4 would be:

```
Name: Ram
Age: 30
Post: TGT
Name: Shyam
Age: 17
Standard: 12
```

12.7 Hybrid inheritance

The combination of two or more forms of inheritance is known as hybrid inheritance. As an example figure 12.9 shows hybrid inheritance which is the combination of multilevel inheritance and multiple inheritance.

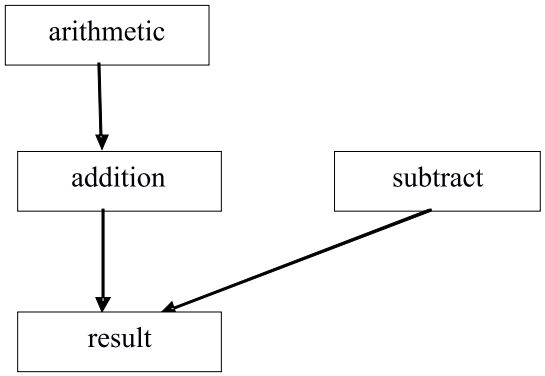


Fig.12.9 An example of hybrid inheritance

Program 12.5: Hybrid inheritance

```
#include<iostream>
using namespace std;
class arithmetic
{
protected:
        int num1, num2;
public:
        void getdata(void)
        {
                cout<<“For Addition:”;
                cout<<“\nEnter the first number: ”;
                cin>>num1;
                cout<<“\nEnter the second number: ”;
                cin>>num2;
        }
};

class addition:public arithmetic
{
protected:
        int sum;
public:
        void add(void)
        {
                sum=num1+num2;
        }
};

class subtract
{
```

```
protected:
        int n1,n2,diff;
public:
        void sub(void)
        {
                cout<<"\nFor Subtraction:";
                cout<<"\nEnter the first number:";
                cin>>n1;
                cout<<"\nEnter the second number:";
                cin>>n2;
                diff=n1-n2;
        }
};
class result:public addition, public subtract
{
public:
        void display(void)
        {
                cout<<"\nSum of "<<num1<<" and "<<num2<<"= "<<sum;
                cout<<"\nDifference of "<<n1<<" and "<<n2<<"= "<<diff;
        }
};

int main()
{
        result z;
        z.getdata();
        z.add();
        z.sub();
        z.display();
        return 0;
}
```

The output of the program 12.5 would be:

```
For Addition:
Enter the first number: 5
Enter the second number: 7
For Subtraction:
Enter the first number: 10
Enter the second number: 3
Sum of 5 and 7 is 12
Difference of 10 and 3 is 7
```

12.8 Virtual Base classes

Consider a hybrid inheritance in which three forms of inheritance which are multilevel, multiple and hierarchical inheritance are involved. This is shown in figure 12.10 . The class 'TA'(Teacher Assistant) has two direct base classes 'teacher' and 'student' which have a common base class 'person'. The class 'TA' inherits the features of 'person' via two different paths. This situation creates a problem that all public and protected members of 'person' are inherited into 'TA' twice, first via 'teacher' and second via 'student'. This form of inheritance creates an ambiguity and should be avoided.

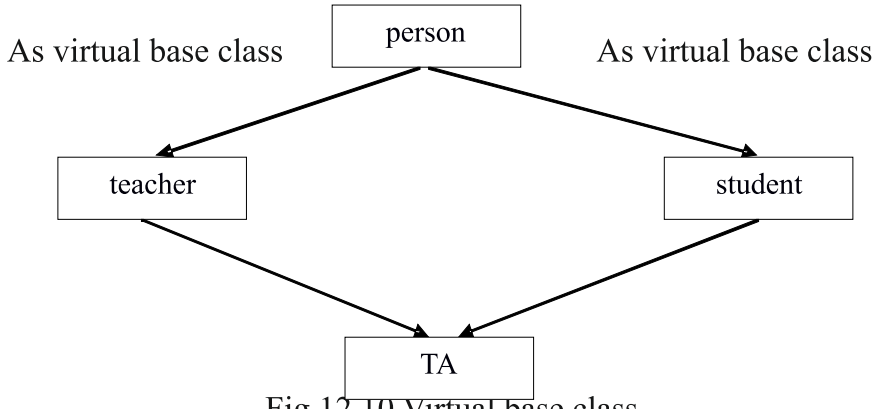


Fig.12.10 Virtual base class

This ambiguity can be resolved by making the common base class as virtual base class while declaring the direct base classes as shown in below.

```
class person
{
   - - - - - - - -
   - - - - - - - -
};
class teacher : virtual public person
{
   - - - - - - - -
   - - - - - - - -
};
class student : virtual public person
{
   - - - - - - - -
   - - - - - - - -
};
class TA: public teacher, public student
```

```
{
   - - - - - - - -
   - - - - - - - -
};
```

When a class is declared as a virtual base class, only one copy of the all public and protected members of that class is inherited.

12.9 Abstract classes

If same function name is used in both base and derived classes, the function in base class is declared as virtual that is called a virtual function. A virtual function with empty body is called pure virtual function. A class having at least one of its member functions as pure virtual function is known as abstract class. It is not used to create objects. It is used only to act as a base class to be inherited by other classes. The pure virtual function must be defined by the class which is derived from the abstract base class. The following program is an example of abstract base class.

Program 12.6: Abstract base class

```
#include <iostream>
using namespace std;
 class Shape
{
protected:
            int width;
            int height;
public:
  virtual int area() = 0;          // pure virtual function
  void getdata(int w, int h)
  {
     width=w;
     height=h;
  }
};

class Rectangle: public Shape
{
public:
  int area()
```

```
  {
    return (width * height);
  }
};
class Triangle: public Shape
 {
public:
  int area() {
    return (width * height)/2;
  }
};
  int main(void)
{
  Rectangle Rect;
  Triangle  Tri;
   Rect.getdata(5,7);
    cout << "Area of Rectangle :" << Rect.area() <<"\n";
  Tri.getdata(6,7);
  cout << "Area of Triangle :"<< Tri.area() <<"\n";
  return 0;
}
```

The output of the program 12.6 would be:
Area of Rectangle: 35
Area of Triangle: 21

Important Points

- The existing classes are used to create new classes, this mechanism is called inheritance.
- By default, the visibility-mode is private.
- In single inheritance, there is one base class and one derived class.
- There can be any number of levels in multilevel inheritance.
- When a class inherits the features of two or more classes, is known as multiple inheritance.
- When a base class is inherited by two or more derived classes, is known as hierarchical inheritance.
- The combination of two or more forms of inheritance is known as hybrid inheritance.
- When a class is declared as a virtual base class, only one copy of the all public and protected members of that class is inherited.

- A class having at least one of its member functions as pure virtual function is known as abstract class.

Practice Questions

Objective type questions:

Q.1 In inheritance, the existing class is called

A. Base class B. Parent class

C. Super class D. All of these

Q.2 In inheritance, the new class is called

A. Derived class B. child class

C. Subclass D. All of these

Q.3 By default, the visibility-mode is

A. Public B. Private

C. Protected D. None of these

Q.4 When there is one base class and one derived class is called

A. Single inheritance B. Multilevel inheritance

C. Multiple inheritance D. Hierarchical inheritance

Q.**5** When a class inherits the features of two or more classes is called

A. Single inheritance B. Multilevel inheritance

C. Multiple inheritance D. Hierarchical inheritance

Very Short Answer Type Questions

Q.1 What is inheritance?

Q.2 What is single inheritance?

Q.3 What is multilevel inheritance?

Q.4 What is multiple inheritance?

Q.5 What is hierarchical inheritance?

Q.6 What is hybrid inheritance?

Q.7 What is abstract class?

Short Answer Type Questions

Q.1 Explain the effect of visibility-mode in inheritance.

Q.2 What is the concept of virtual base class?

Essay Type Questions

Q.1 Write a program to create an abstract class 'shape' consist of a pure virtual function 'volume'. The 'shape' class is inherited by three classes called 'cone', 'cylinder' and 'cube', these derived classes define the pure virtual function 'volume' to compute the volume.

Answer Key

1. D 2. D 3. B 4. A 5. C