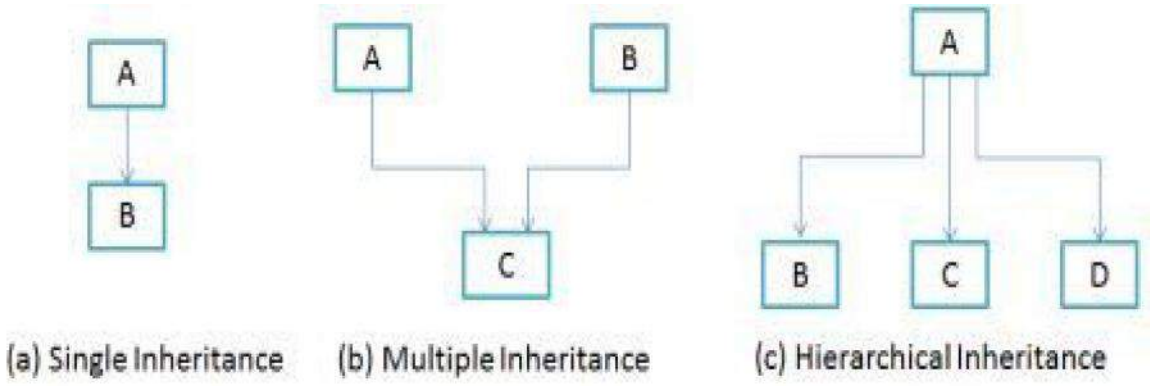
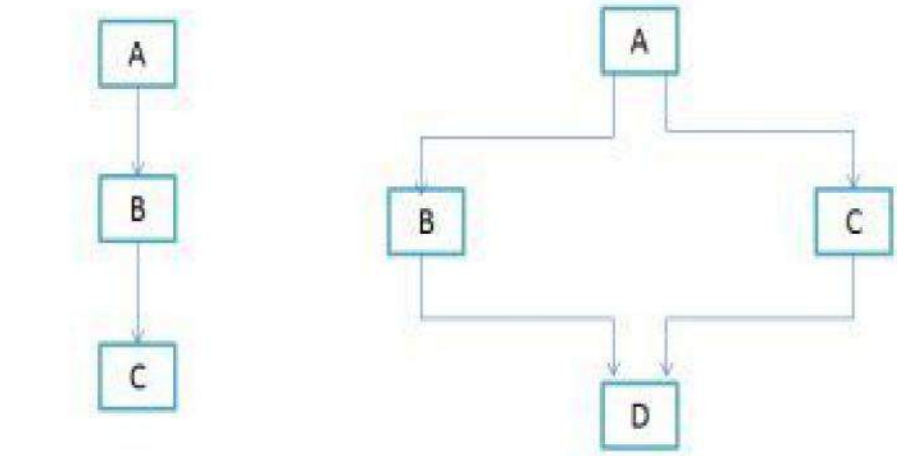
# CHAPTER - 10
# INHERITANCE

## OBJECTIVES

- **To understand the concepts of inheritance.**
- **Usage of inheritance.**
- **The role of visibility mode.**
- **Levels of inheritance.**
- **Concepts of virtual base class.**
- **Concepts of abstract class.**
- **Constructors and destructors in derived class.**




(a) Single Inheritance (b) Multiple Inheritance (c) Hierarchical Inheritance




(d) Multilevel Inheritance (e) Hybrid Inheritance

## 10.1 Introduction:

Inheritance is another important aspect of object oriented programming. C++ supports this concept. C++ classes can be used in several ways. This is basically done by creating new classes, reusing the properties of existing one.

C++ allows the user to create a new class (derived class) from an existing class (base class). The derived class inherits all features from a base class and it can have additional features of its own.

> Inheritance is the capability of one class to inherit properties from another class.

## 10.2 Base Class:

It is the class whose properties are inherited by another class. It is also called Super Class.

## 10.3 Derived Class:

It is the class that inherits properties from base class (es).It is also called Sub Class.

Inheritance has following advantages:

1. Reusing existing code
2. Faster development time
3. Easy to maintain
4. Easy to extend
5. Memory utilization

The following example illustrates the needs of inheritance.

Suppose X is a class already defined and we need to redefine another class Y having same properties of X and in addition to its own. Suppose if we use direct option without using inheritance, it has following problems.

1. Code written in X is repeated again in Y which leads to unnecessary wastage of memory space.
2. Testing to be done separately for both class X and class Y leads to wastage of time.

The above problems can be solved by using the concept of inheritance.

If we reuse the code of X even in Y without rewriting it. The class Y inherits all the properties of X. The class X is called Base class and the class Y is called derived class.

### 10.3.1 Defining derived classes

When you declare a class, you can indicate what class it derives from by writing a colon after the class name, the type of derivation (public or private or protected) and the class from which it derives.

```
class derived_class_name : visibility_mode base_class_name
{
        // .....
        //members of the derived class
};
```

Where, class → keyword
derived_class_name → Name of the derived class
: → shows the derivation from the base class
visibility mode → Specifies the type of derivation
base_class → Name of the base class

The body of the derived class contains its own data members and member function. Like base class definition, the derived class definition must be terminated by a semicolon.

If no visibility mode is specified, then by default the visibility mode is considered private.

Note that all members of the class except private are inherited. Following are some examples of derived class definition.

### 10.3.2 public derived class

```
class  father        // Base class
{
  private:
        char    name[50];
        int      age;
  public:
        char    caste[50];
        int     boys;
        int     girls;
        void    readdata();
        void    printdata();
};
class son: public father                    //public derived class
{
  private:
        char company[50];
        float  salary;
  public:
        void getdata();
        void display();
};
```

Similarly we can write private derived class.

### 10.3.3 private derived class

```
class son: private father          // private derived class
    {
      private:
              char    company[50];
              float   salary;
      public :
              void    getdata();
              void    display();
    };
```

Similarly we can write protected derived class as:

### 10.3.4 protected derived class

```
    class son: protected  father        // protected  derived class
    {
      private:
              char    company[50];
              float   salary;
      public :
              void    getdata();
              void    display();
    };
```

## 10.4 Visibility mode

The visibility mode (private, public and protected) in the definition of the derived class specifies whether features of the base class are privately derived or publicly derived or protectedly derived. The visibility mode basically controls the access specifier to be for inheritable member of base class in the derived class.

The role of visibility modes:

| Base class | Derived class | | |
|---|---|---|---|
| | Public mode | Private mode | Protected mode |
| Private | Not inherited | Not inherited | Not inherited |
| Public | Public | Private | Private |
| Protected | Protected | Private | Private |

### 10.4.1 Public Inheritance

This is the most used inheritance mode. In this

- The public members of a base class become public members of the derived class.

- The private members of a base class can not be inherited to the derived class.
- The protected members of a base class stay protected in a derived class.

class subclass: **public** superclass

```
class student                          //base class
{
        private:
                int     rollno;
                char    name[50];
                float   per;
        public:
                void    input();
                void    output();
};
class comp: public student             //publicly derived class
{
        private:
                int      marks;
        public:
                void    read();
                  void write();
};
```

### 10.4.2. Private Inheritance

- The public members of a base class become the private members of the derived class.
- The private members of a base class cannot be inherited to the derived class.
- The protected members of a base class stay protected in a derived class.

### 10.4.3. Protected Inheritance

- The public members of a base class become protected in a derived class.
- The private members of a base class cannot be inherited to the derived class.
- The protected members of a base class stay protected in a derived class.

class subclass : **protected** superclass

## 10.5 Levels of Inheritance

A derived class extends its features by inheriting some or all the properties from its base class and adding new features of its own. While inheriting, the derived class can share properties from:

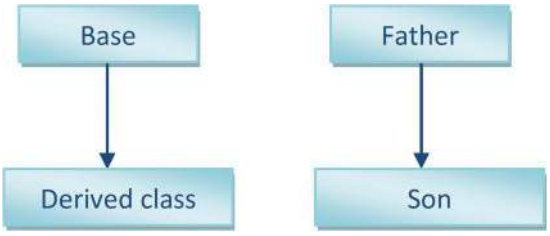
- Only one class
- More than one class
- More than one level

Based on this relationship, inheritance can be classified into five forms.

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

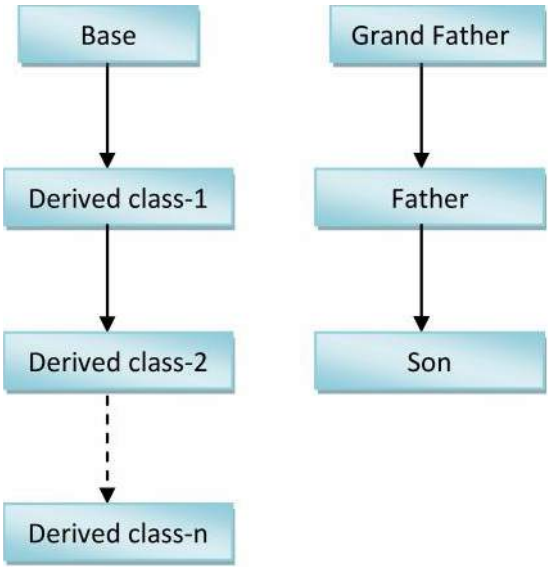
### 10.5.1 Single Inheritance

If a class is derived from a single base class, it is called as single inheritance.



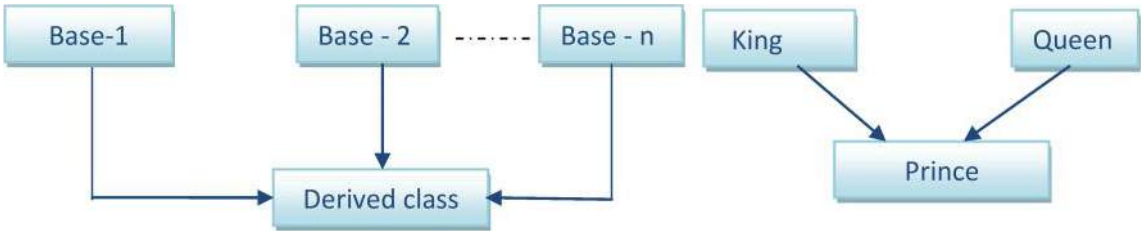

### 10.5.2 Multilevel Inheritance

The classes can also be derived from the classes that are already derived. This type of inheritance is called multilevel inheritance.



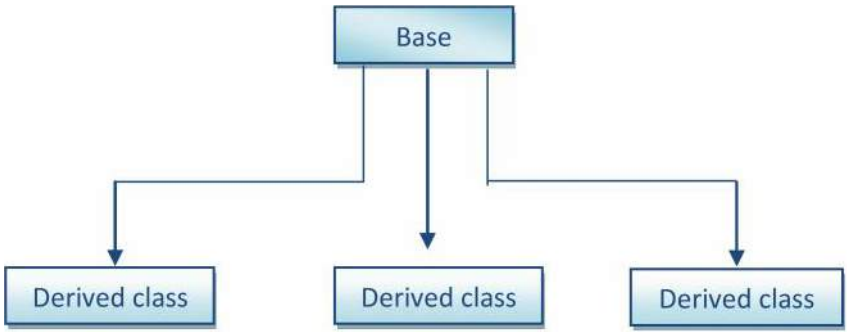
### 10.5.3 Multiple Inheritance

If a class is derived from more than one base class, it is known as multiple inheritance.

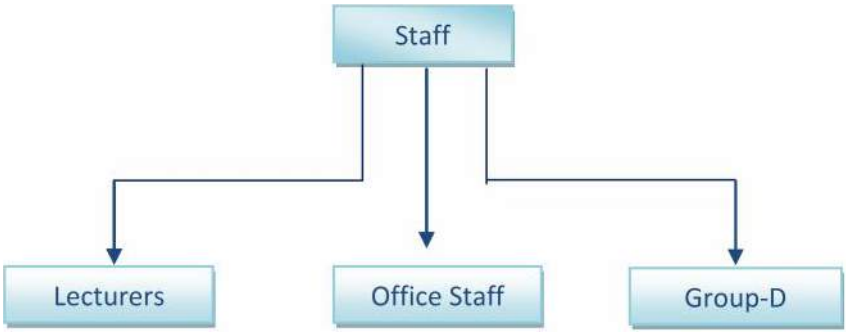



### 10.5.4 Hierarchical Inheritance

If a number of classes are derived from a single base class, it is called as hierarchical inheritance.

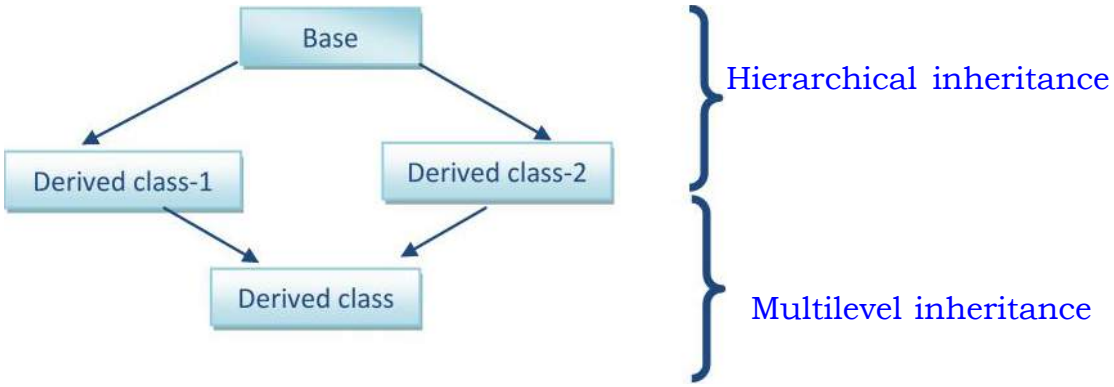



Example for Hierarchical Inheritance




### 10.5.5 Hybrid Inheritance

Hybrid Inheritance is combination of Hierarchical and Multilevel Inheritance.




**Program to illustrate single level inheritance**

```
#include<iostream.h>
#include<conio.h>
class base
{
        private:
                int     rollno;
                char    name[20];
public:
                void read()
                {
                        cout<<"Enter Roll No and name "<<endl;
                        cin>> rollno >>name;
                }

                void display()
                {
                        cout<<"roll no: "<< rollno <<endl;
                        cout<<"name :  "<<name <<endl;
                }

};
class derive: public base
{
        private:
                int     m1;
                int     m2;
                int     t;
        public:
        void read1()
        {
                cout<<"enter first marks and second marks: "<<endl;
```

```
                cin>>m1>>m2;
                t = m1+m2;
        }
        void display1()
        {
                cout<<"First marks = "<<m1<<endl;
                cout<<"Second marks =  "<<m2 <<endl;
                cout<<"Total marks =  "<<t<<endl;
        }

 };
void  main()
{
        derive  ob;
        clrscr();
        ob.read();
        ob.read1();
        ob.display();
        ob.display1();
        getch();
}
```

```
Enter Roll No and name
1234  Ravindra
Enter first marks and second marks:  100  100
Roll no : 1234
Name :   Ravindra
First marks =  100
Second marks = 100
Total marks = 200
```

## 10.6 Relationship between classes

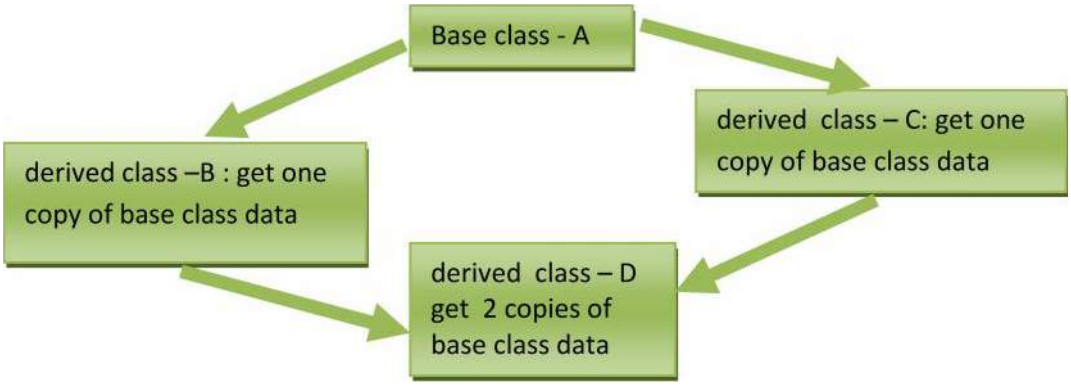
### 10.6.1. Virtual base classes:

Consider a situation where the program design would require one base class (call it A) and two derived classes namely B and C, which are inherited from the base class A. Further, derived class D is created from B and C. See the figure.

In the public inheritance, B and C inherit one copy of the base class data, where as derived class D inherited from B and C get two copies of the base class data.

Now suppose a member function of D now wants to access the data members of the base class an ambiguity problem of which of the two copies is to access will

be encountered. The compiler will generate an error message for this ambiguity. To overcome this problem, the derived class B and C should be declared as virtual.




When two or more objects are derived from a common base class, we can prevent multiple copies of the base class being present in an object derived from those objects by declaring the base class as virtual when it is being inherited. Such a base class is known as virtual base class. This can be achieved by preceding the base class name with the word **virtual.**

Example:

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

### 10.6.2 Abstract classes

An abstract class is one that is not used to create objects. An abstract class is designed only to act as a base class (to be inherited by other classes). It is a design concept in program development and provides a base upon which other classes may be built. In the previous example, the class A is an abstract class since it was not used to create any objects.

### 10.6.3 Constructors in derived classes

As we know, the constructors play an important role in initializing objects. We did not use them earlier in the derived classes for the sake of simplicity. One important thing to note here is that, as long as no base class constructor takes any arguments, the derived class need not have a constructor function. However, if any base class contains a constructor with one or more arguments, then it is mandatory for the derived class to have a constructor and pass the arguments to the base class constructors. Remember, while applying inheritance we usually create objects using the derived class. Thus, it makes sense for the derived class to pass arguments to the base class constructor. When both the derived and base classes contain constructors, the base constructor is executed first and then the constructor in the derived class is executed.

In case of multiple inheritances, the base classes are constructed in the order in which they appear in the declaration of the derived class. Similarly, in a multilevel inheritance, the constructors will be executed in the order of inheritance.

Since the derived class takes the responsibility of supplying initial values to its base classes, we supply the initial values that are required by all the classes together, when a derived class object is declared. How are they passed to the base class constructors so that they can do their job? C++ supports a special argument passing mechanism for such situations.

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

Example: Use of constructor in single inheritance

```
class  base_class
{
        protected:
        public:
                base_class()          // base class constructor
                {
                }
};
```

```
class  derived_class: public base_class
{
        protected:
        public :
                derived _class()     // derived  class constructor
                {
                }
};
```

### 10.6.4 Destructors in derived classes

If the constructors are called down the line from the base to the derived class, the destructors are called just in the reverse order. That is from the derived class up to the base class.

## Points to remember:

- Inheritance: It is the capability of one class to inherit properties from another class.
- Base Class: It is the class whose properties are inherited by another class. It is also called Super Class.
- Derived Class: It is the class that inherits properties from base class(es). It is also called Sub Class.
- Advantages of inheritance
  - Reusing existing code.
  - Faster development time.
  - Easy to maintain.
  - Easy to extend.
  - Memory utilization
- All members of the class except private are inherited.
- The visibility mode (private, public protected) in the definition of the derived class specifies where features of the base class are privately derived or publicly derived or protected derived. The visibility mode basically controls the access specifier to be for inheritable member of base class in the derived class.
- Inheritance can be classified into five forms.
  - Single inheritance
  - Multilevel inheritance
  - Multiple inheritance
  - Hierarchical inheritance
  - Hybrid inheritance

- Single Inheritance: If a class is derived from a single base class, it is called as Single Inheritance.
- Multilevel Inheritance: The classes can also be derived from the classes that are already derived. This type of inheritance is called multilevel inheritance.
- Multiple Inheritance: If a class is derived from more than one base class, it is known as multiple inheritance.
- Hierarchical Inheritance: If a number of classes are derived from a single base class, it is called as hierarchical inheritance.
- Hybrid inheritance: Hybrid Inheritance is combination of Hierarchical and Multilevel Inheritance.
- In the publicly derived class, the public and protected members remain public and protected.
- In the privately derived class, the public and protected members of the base class become private members.
- In the protected derived class, the public and protected members of the base class become protected members.
- Virtual base class: When two or more objects are derived from a common base class, we can prevent multiple copies of the base class being present in an object derived from those objects by declaring the base class as virtual when it is being inherited. Such a base class is known as virtual base class. This can be achieved by preceding the base class' name with the word **virtual.**
- An abstract class is one that is not used to create objects. An abstract class is designed only to act as a base class (to be inherited by other classes).

## Review questions

### One marks questions:

1. What is inheritance?
2. How to implement inheritance?
3. What is base class?
4. What is derived class?
5. What is public access specifier?
6. What is private access specifier
7. Mention any one advantage of inheritance?
8. Is inheritance possible in c?
9. What is the use of protected access specifier?
10. What is the use of public access specifier?
11. What is the use of private access specifier?
12. What is singe inheritance?
13. What is multilevel inheritance?
14. What is hierarchical inheritance?
15. What is hybrid inheritance?
16. What is multiple inheritance?
17. What is virtual base class?
18. What is an abstract class?
19. When is it necessary to use inheritance?
20. What is visibility mode?

### Two marks questions:

1. How to implement inheritance?
2. What is the difference between public and private access specifier?
3. Mention any 2 advantages of inheritance?
4. Mention any 2 types of inheritance?
5. What is the difference between inheritance and polymorphism?
6. What is singe inheritance? Give an example.
7. What is multilevel inheritance? Give an example.
8. What is hierarchical inheritance? Give an example.
9. What is hybrid inheritance? Give an example.
10. What is multiple inheritance? Give an example.
11. What is virtual base class? Give example.
12. What is an abstract class?
13. Which are the components which cannot be inherited?
14. Explain Single Inheritance with a suitable C++ program.
15. Explain the requirements of a virtual base class.
16. When is it necessary to use inheritance?
17. What is visibility mode? What is its role?

**18.** How does inheritance influence the working of constructors?

**19.** How does inheritance influence the working of destructors?

**Three marks questions:**

**1.** What is the difference between public and private access specifier with respect to inheritance?

**2.** What are the advantages of inheritance?

**3.** What are the types of inheritance?

**Five marks questions:**

**1.** What is the difference between public and private and protected access specifier?

**2.** What are the advantages of inheritance?

**3.** What are the types of inheritance? Explain any 2.

**4.** What is virtual base class? Give example.

5. Which are the components which can not be inherited?

**6.** Explain singe inheritance with a suitable C++ program.

**7.** Explain the requirements of a virtual base class.

**8.** What is visibility mode? What is its role with respect to inheritance?

**9.** How does inheritance influence the working of constructors and destructors?

******