Chapter 6:

INHERITANCE

5hours.

Inheritance:

- Inheritance is the process by which objects of one class acquires the properties of another class.
- The technique of building new classes from the existing classes is called **Inheritance**.
- Inheritance is one of the most important feature of Object Oriented Programming which provides code reusability.
- The idea of inheritance implements the **is a** relationship . For e.g. Orange is a Fruit.

Syntax:

```
 class Base_class {
//body of base_class
};
  class Derived_class : Visibility_Mode  Base_class
{
//body of derived_class
};
```

E.g.

```
class vehicle {
 body of this class;
  };
class Bus : public vehicle{
 body of this class;
  };
```

Q. Why to use Inheritance?

Ans: Inheritance is used for following :

1.It helps in reusability of the code that are already exist in our program which saves time , effort and cost of the program development process.

2. It helps in development of reliable and less error prone software or program.

Base And Derived Class:

- **Base Class :** The class whose properties are inherited by sub class or derived class is called Base Class or Super class or parent class. In other word , the existing classes that are used to derive new classes are called as base class.
- **Derived Class :** The class that inherits properties from another class (or base class) is called Derived Class or Sub class or child class . In other word , the new classes derived from existing classes are called derived class.

protected Access Specifier:

- **protected** access specifier are useful in inheritance.
- These specifier are visible by the member function and friend function of a class where it is declared.
- The members declared as **protected** are accessible from outside the class BUT only in a class derived from it.
- Any data members that the derived class might need to access should be made **protected** rather than **private**. That ensure the class is ready for inheritance.
- But the **protected** members are less secure than the **private** members as **protected** members are easily accessible to the derived class.

Do Yourself : Advantage and Disadvantage of using protected access specifier in inheritance?

Derived Class Declaration:

1. public Derivation:

```
class derived : public base          //public is visibility mode or access specifier.
{
    //members
};
```

2. protected Derivation:

```
class derived : protected base      //protected is visibility mode or access specifier.
{
  //members
};
```

3. private Derivation:

class derived : private base //private is visibility mode or access specifier.

```
{
 //members
};
```

- If no access specifier is given then it is by default private access specifier for class.

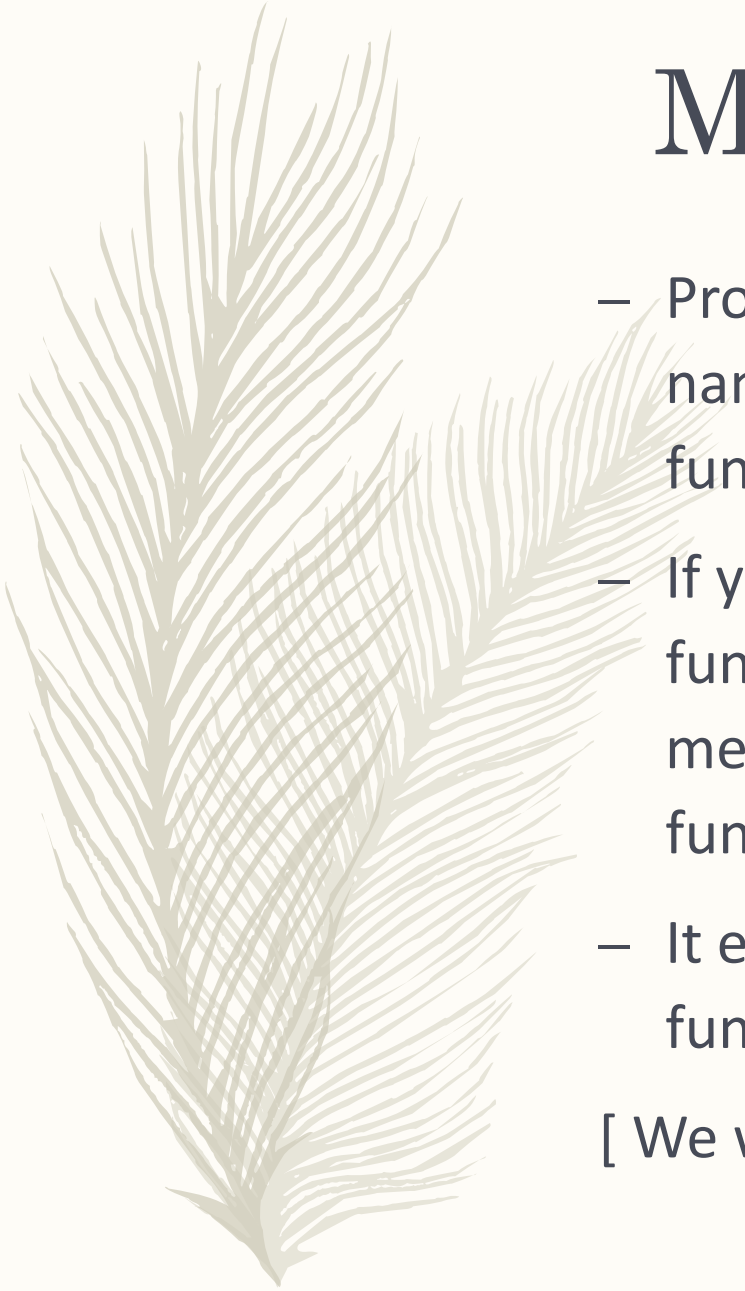
```
class derived : base    // here, by default private access specifier.
{
//members
};
```

Inheritance Visibility Mode :-

Members in Base class	Derived class		
	Public mode	Private mode	Protected mode
Private	Not accessible	Not accessible	Not accessible.
Protected	Protected	Private	Protected.
Public.	Public	Private	Protected.

Accessibility check :-

Access	Public	Protected	Private.
Same class	Yes	Yes	Yes.
Derived class	Yes	Yes	No.
Outside class	Yes	No	No.



Member Function Overriding:

- Process of creating member function in derived class with same name (or you can say same signature) as that of visible member functions of base class is known as member function overriding.
- If you creates an object of the derived class and call the member function which exist in both classes(base and derived), the member function of derived class is invoked and the member function of base class is ignored.
- It enables us to provide specific implementation of the member function which is already provided by its base class.

[We will program it on IDE for our understanding.]

Types/Forms of Inheritance:

Basically, there are following types of inheritance:

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

[Multipath inheritance is one of its forms too.]

1. Single Inheritance:

– In this type of inheritance, **one derived class** inherits from only **one base class**. It is the most simplest form of Inheritance.

Syntax:

```
class Base{
//body of base class
};
class Derived: access_specifier  Base
{
//body of Derived class.

};
```

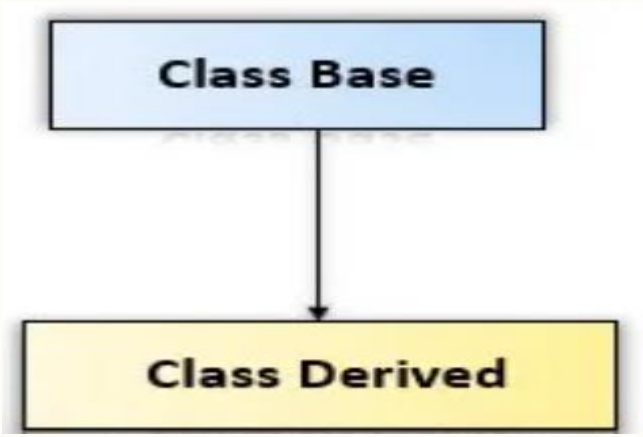


Fig: Single Inheritance.

[Here, access_specifier can be public, private or protected]

2. Multiple Inheritance:

– In this type of inheritance, **a single derived class** may inherit from **two or more than two base classes**.

Syntax:

```
class Base1{
//body of base class
};
class Base2{
//body of base class
};
class Derived : access_specifier  Base1, access_specifier Base2
{
//body of Derived class.
};
```

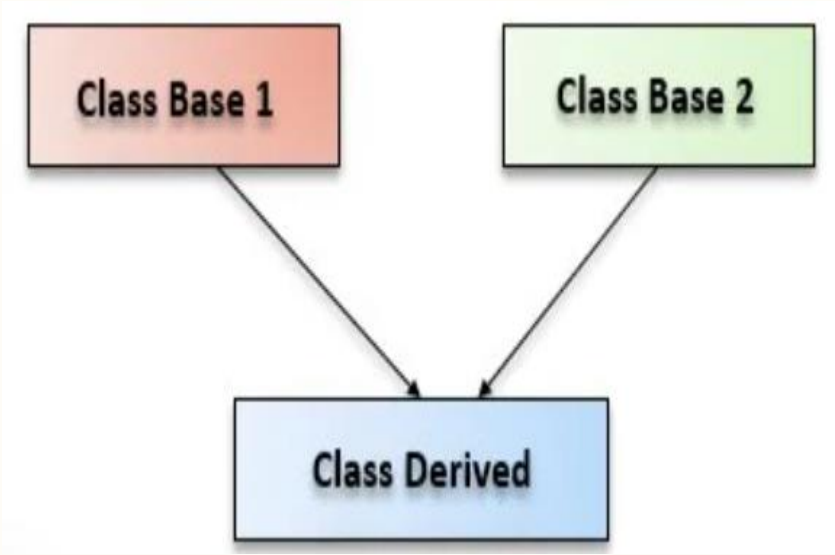


Fig: Multiple Inheritance.

3. Multilevel Inheritance:

- In this type of inheritance ,the **derived class** inherits from a **class**, which in turn inherits from **some other class**. The **base class** for one, is **derived class** for the other.

Syntax:

```
class Base{
//body of base class
};
class Derived1: access_specifier Base
{
//body of Derived1 class
};
Class Derived2: access_specifier Derived1
{
//body of Derived2 class
};
```

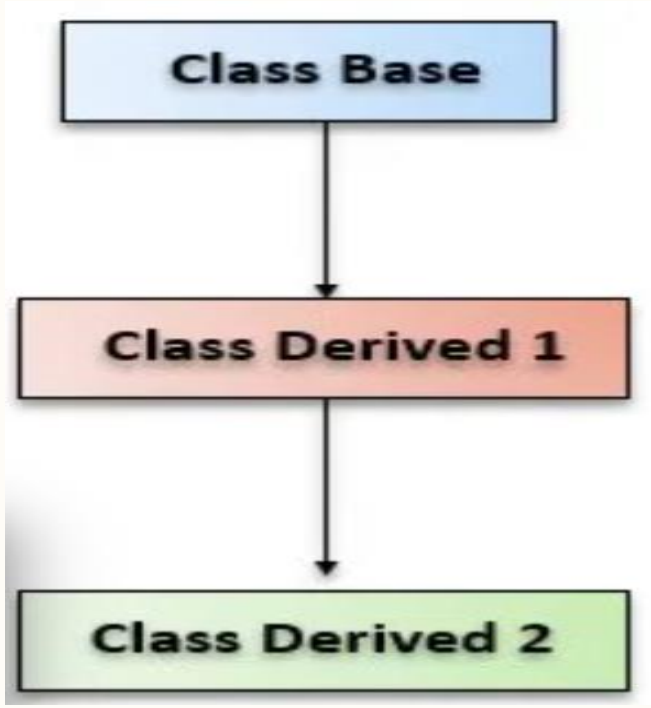


Fig: Multilevel Inheritance

4. Hierarchical Inheritance:

In this type of inheritance, **multiple derived classes** inherits from a **single base class**.

Syntax:

```
class Base{
//body of base class
};
class Derived1: access_specifier  Base
{
//body of Derived1 class
};
Class Derived2: access_specifier  Base
{
//body of Derived2 class
};
```

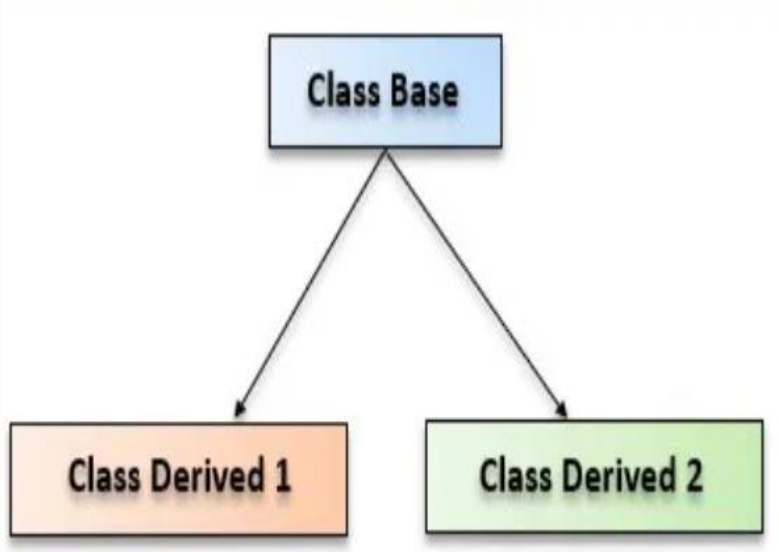


Fig: Hierarchical Inheritance

5. Hybrid Inheritance:

Hybrid Inheritance is the combination of two or more inheritances .

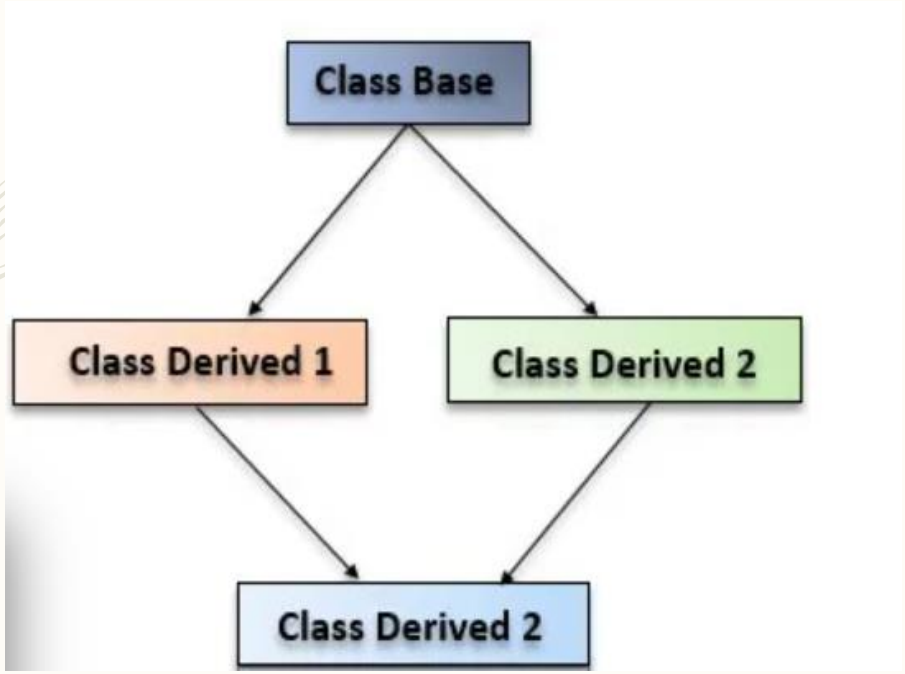
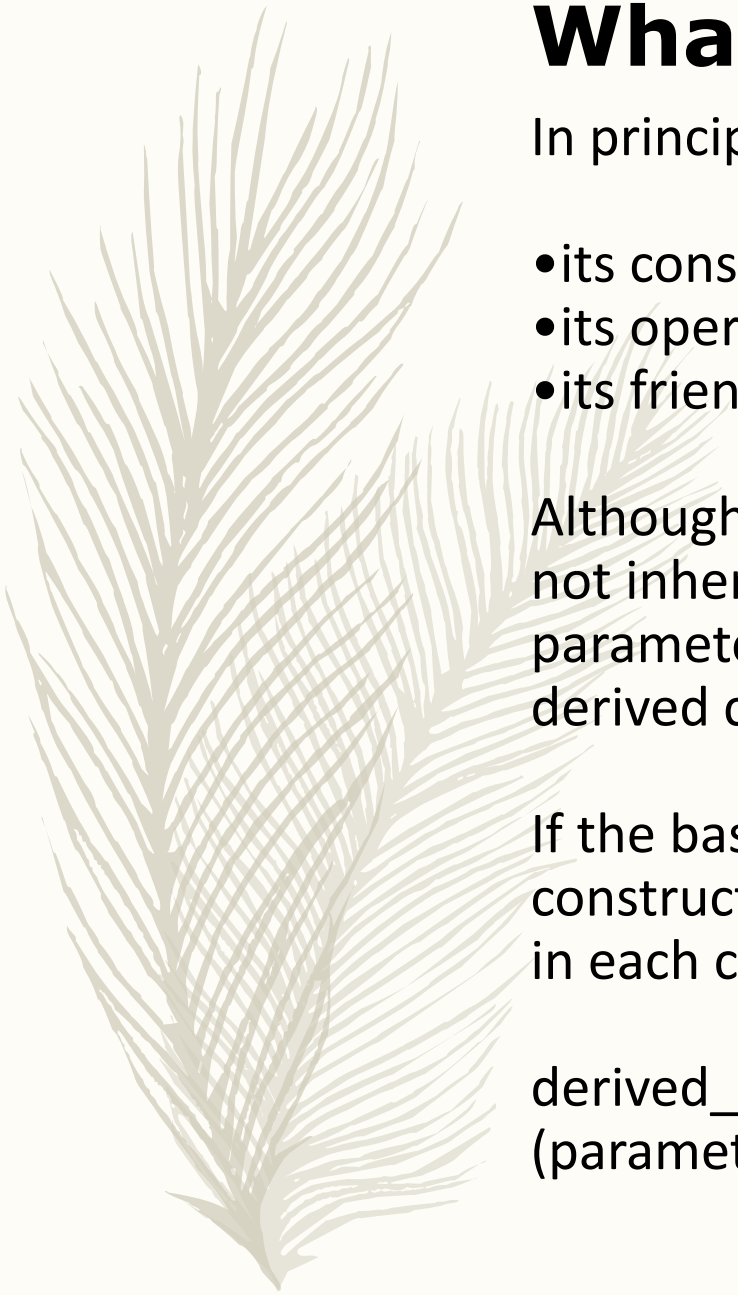


Fig: Hybrid Inheritance.

[**Your Work:** what is the code representation(syntax only) for above diagram??]



What is inherited from the base class?

In principle, a derived class inherits every member of a base class except:

- its constructor and its destructor
- its operator=() members
- its friends

Although the constructors and destructors of the base class are not inherited themselves, its default constructor (i.e., its constructor with no parameters) and its destructor are always called when a new object of a derived class is created or destroyed.

If the base class has no default constructor or you want that an overloaded constructor is called when a new derived object is created, you can specify it in each constructor definition of the derived class:

derived_constructor_name (parameters) : base_constructor_name (parameters) {...}

Initializer List in C++

- Initializer list is used to initialize data members of class.
- The list of members to be initialized is indicated with constructor as a comma-separated list followed by a colon.
- The initializer list does not end in a semicolon.
- **Syntax:**

Constructor_Name (datatype value1,datatype value2): data_member(value1), data_member(value2)

{ //body }

We will program for all these in codeblock …..

[**Refer book for theory section of following topics.]

- Multipath Inheritance and Virtual Base Class.
- Constructor Invocation in Single and Multiple Inheritances.
- Destructor in Single and Multiple Inheritances.