C++ Inheritance

One of the most important concepts in object-oriented programming is that of inheritance. Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time.

When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the **base** class, and the new class is referred to as the **derived** class.

C++ Inheritance

Inheritance is the process by which new classes called *derived* classes are created from existing classes called *base* classes.

The derived classes have all the features of the base class and the programmer can choose to add new features specific to the newly created derived class.

C++ Inheritance

Features or Advantages of Inheritance:

Reusability:

✓Inheritance helps the code to be reused in many situations.

✓The base class is defined and once it is compiled, it need not be reworked.

✓Using the concept of inheritance, the programmer can create as many derived classes from the base class as needed while adding specific features to each derived class as needed.

C++ Inheritance

Features or Advantages of Inheritance:

✓***Saves Time and Effort:***

The above concept of reusability achieved by inheritance saves the programmer time and effort. The main code written can be reused in various situations as needed.

✓**Increases Program Structure which results in greater reliability.**

C++ Inheritance

General Format for implementing the concept of Inheritance:

class derived_classname: access specifier baseclassname

For example, if the *base* class is *MyClass* and the derived class is sample it is specified as:

class sample: public MyClass

The above makes sample have access to both *public* and *protected* variables of base class *MyClass*

C++ Inheritance

Reminder about public, private and protected access specifiers:

1 If a member or variables defined in a class is private, then they are accessible by members of the same class only and cannot be accessed from outside the class.

2 Public members and variables are accessible from outside the class.

3 Protected access specifier is a stage between private and public. If a member functions or variables defined in a class are protected, then they cannot be accessed from outside the class but can be accessed from the derived class.

Type of Inheritance

When deriving a class from a base class, the base class may be inherited through **public, protected** or **private** inheritance. The type of inheritance is specified by the access- specifier.

We hardly use **protected** or **private** inheritance, but **public** inheritance is commonly used. While using different type of inheritance, following rules are applied:

Type of Inheritance

✓**Public Inheritance:** When deriving a class from a **public** base class, **public** members of the base class become **public** members of the derived class and **protected** members of the base class become **protected** members of the derived class. A base class's **private** members are never accessible directly from a derived class, but can be accessed through calls to the **public** and **protected** members of the base class.

Type of Inheritance

✓**Protected Inheritance:** When deriving from a **protected** base class, **public** and **protected** members of the base class become **protected** members of the derived class.

✓**Private Inheritance:** When deriving from a **private** base class, **public** and **protected** members of the base class become **private** members of the derived class

C++ Inheritance

Inheritance Example:

```
class MyClass

{       public:
        MyClass(void) { x=0; }
        void f(int n1)
        {  x= n1*5;}
        void output(void) { cout<<x; }
        private:
        int x;
};
```

C++ Inheritance

Inheritance Example:

```
class sample: public MyClass
        { public:
        sample(void) { s1=0; }
        void f1(int n1)
                { s1=n1*10;}
        void output(void)
        { MyClass::output();   cout << s1; }
        private:
        int s1;
        };
```

C++ Inheritance

Inheritance Example:

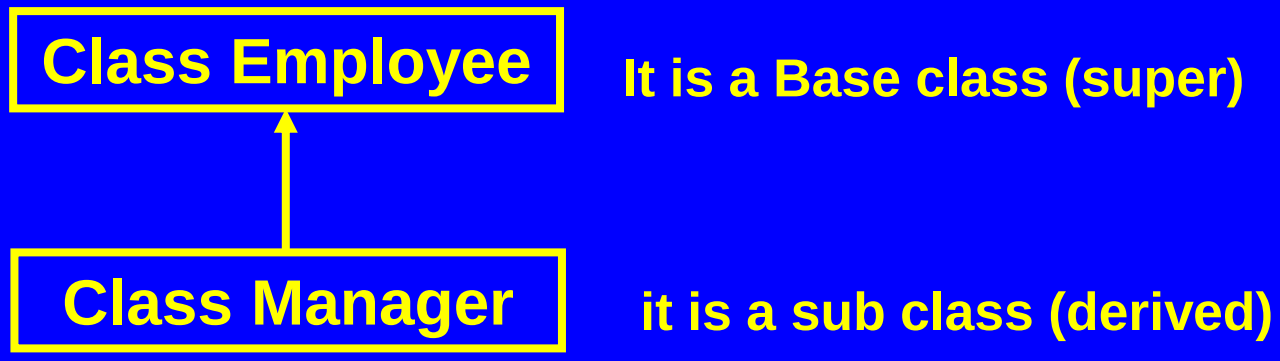
```
int main(void)
 {      sample s;
        s.f(10);
        s.output();
        s.f1(20);
        s.output();
}
```

The output of the above program is
50
200

Types of Inheritance

1. Single class Inheritance:

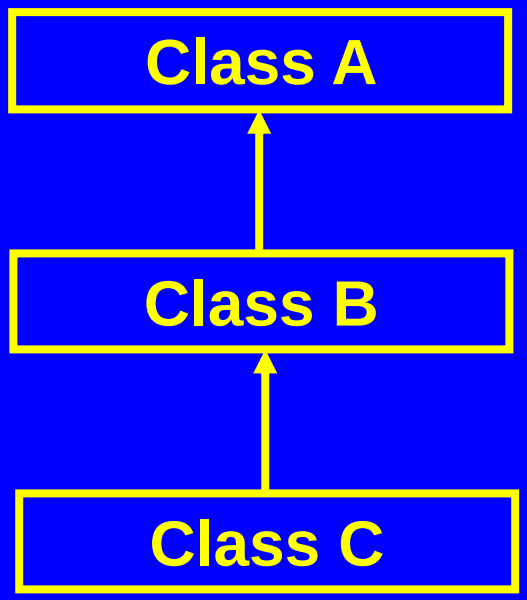
Single inheritance is the one where you have a single base class and a single derived class.



Types of Inheritance

2. Multilevel Inheritance:

In Multi level inheritance, a class inherits its properties from another derived class.



it is a Base class (super) of B

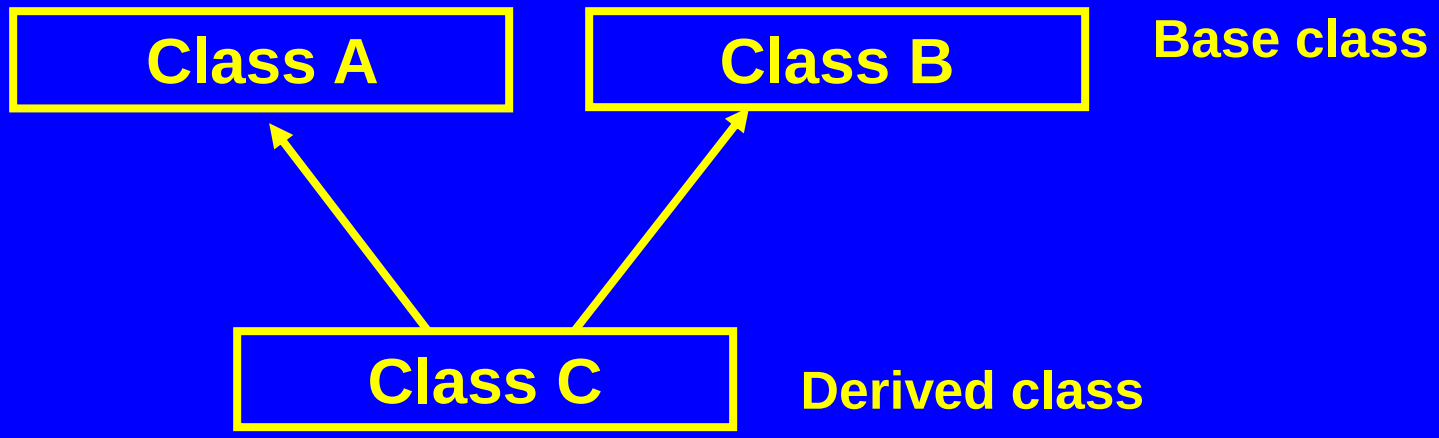
it is a sub class (derived) of A and base class of class C

derived class(sub) of class B

Types of Inheritance

3. Multiple Inheritances:

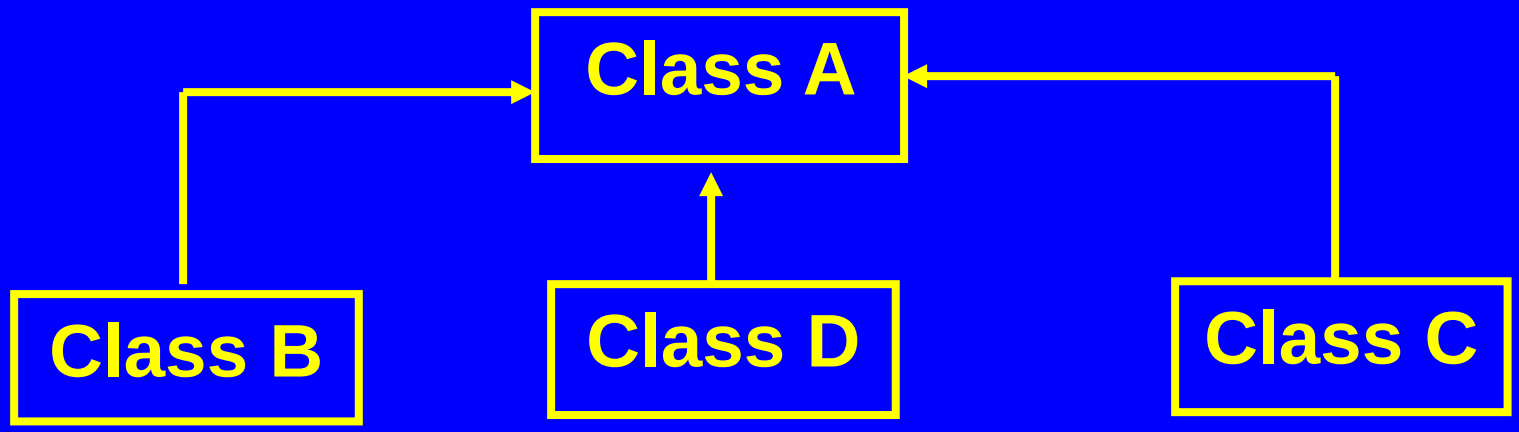
In Multiple inheritances, a derived class inherits from multiple base classes. It has properties of both the base classes.



Types of Inheritance

4. Hierarchical Inheritance:

In hierarchical Inheritance, it's like an inverted tree. So multiple classes inherit from a single base class. It's quite analogous to the File system in a unix based system.



Types of Inheritance

5. Hybrid Inheritance:

✓In this type of inheritance, we can have mixture of number of inheritances but this can generate an error of using same name function from no of classes, which will bother the compiler to how to use the functions.

✓Therefore, it will generate errors in the program. This has known as ambiguity or duplicity.

✓Ambiguity problem can be solved by using **virtual base classes**

Types of Inheritance

5. Hybrid Inheritance: