INHERITANCE IN C++

OVERVIEW

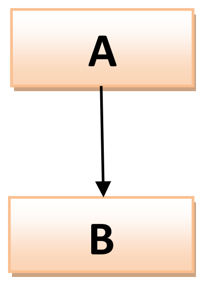
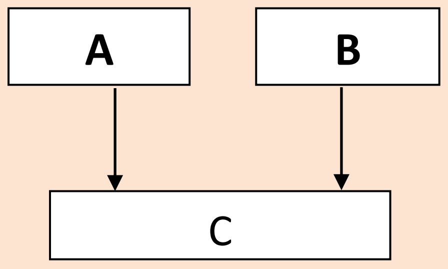
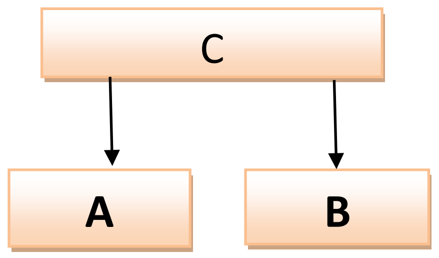
- Reusability is a very important feature of OOPs
- In C++ we can reuse a class and add additional features to it
- Reusing classes saves time and money
- Reusing already tested and debugged class will save a lot of effort of developing and debugging the same thing again.

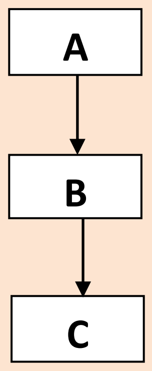
WHAT IS INHERITANCE IN C++

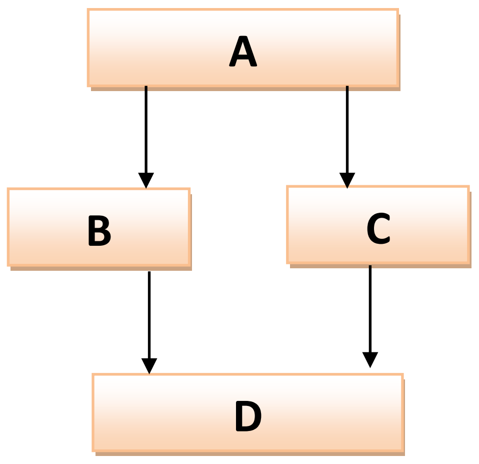
- The concept of Reusability in C++ is supported using Inheritance.
- We can reuse the properties of an existing class by inheriting from it.
- The existing class is called as the *Base* Class.
- The new class which is inherited is called as the *Derived* Class.
- There are different types of inheritance in C++

FORMS OF INHERITANCE IN C++

- Single Inheritance
- Multiple Inheritance
- Hierarchical Inheritance
- Multilevel Inheritance
- Hybrid Inheritance

Single Inheritance	A derived class with only one base class	
Multiple Inheritance	A derived class with more than one base class	
Hierarchical Inheritance	Several derived class from a single base class	

Multilevel Inheritance	Deriving a class from already existing class.
	`1. A is the base class for B and B is the base class for C`
	`2. A-->B-->C is called Inheritance Path`
		

Hybrid Inheritance	• Hybrid inheritance is a combination of multiple inheritance and multilevel inheritance. • A class is derived from two classes as in multiple inheritance. • However, one of the parent classes is not a base class.	

Derived Class syntax:

```
class {{derived-class-name}} : {{visibility-mode}}
{{base-class-name}}
   {
       class members/methods/etc...
   }
```

Visibility Mode:-

1. Public Visibility Mode: Public members of the base class becomes Public members of the derived class
2. Private Visibility Mode: Public members of the base class becomes Private members of the derived class
3. Default visibility mode is private
4. Private members are never inherited but they can be accessed by derived class and its objects *INDIRECTLY* by some public method of base class. They can't be accessed directly.

Important Points:-

→ Constructor of base class gets invoked when an object of derived class is created.
→ Object of the derived class can also have the private properties of base class with the help of some public method of base class.
→ Object of derived class can't call the methods of base class if the visibility mode of derived class is private. Those methods can be invoked by calling them under the definition of some public method of derived class.

Protected Access Modifier :-

Sometimes we want to create a class having members which can't be changed outside the class but they can be inherited by some derived class. In that case, we define those members under protected access modifier.

	Public derivation	Private Derivation	Protected Derivation
1.Private members	**Not Inherited**	**Not Inherited**	**Not Inherited**
2.Protected members	**Protected**	**Private**	**Protected**
3. Public members	**Public**	**Private**	**Protected**

- Protected members of a derived class become protected members of a class derived in a public mode from base class.

Syntax for inheriting in multiple inheritance

```
class DerivedC: visibility-mode base1,visibility-mode base2
{
      Class body of class "DerivedC"
};
```

E.g.

```
class Base1{
protected:
    int base1int;
public:
    void set_base1int(int
a)
    {
        base1int = a;
    }
```

```
class Base2{
protected:
    int base2int;
public:
    void set_base2int(int
a)
    {
        base2int = a;
    }
```

Note:- There can be more than two base class

```
class Derived : public Base1, public Base2, public Base3
{
  public:
      void show(){
          cout << "The value of Base1 is " << base1int<<endl;
          cout << "The value of Base2 is " << base2int<<endl;
          cout << "The value of Base3 is " << base3int<<endl;
          cout << "The sum of these values is " << base1int+
                                 base2int + base3int << endl;}
};
```

→ The inherited derived class will look something like this:

```
Data members:
     base1int --> protected
     base2int --> protected

Member functions:
     set_base1int() --> public
     set_base2int() --> public
     show() --> public
```

Ambiguity Resolution in Inheritance

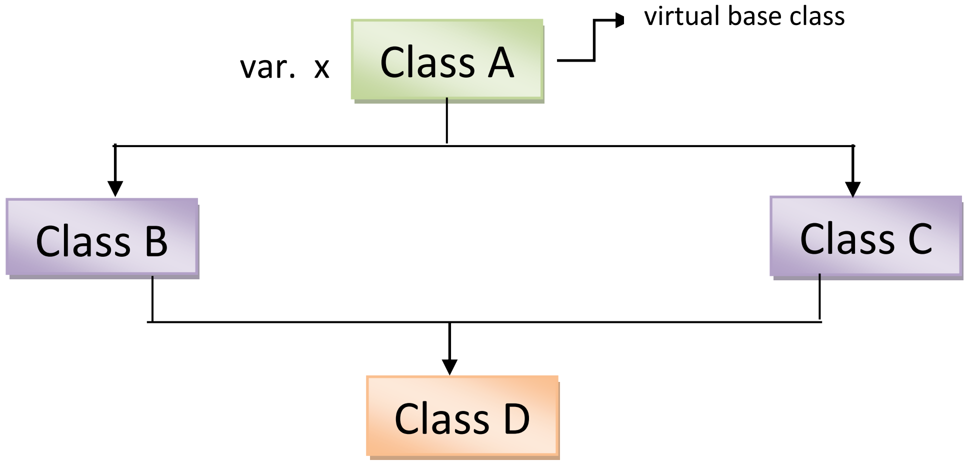
1. If a class is derived publically from two base class and a function of greet() is defined in both the base class, then using greet function for an object of derived class will create an ambiguity. We can resolve this ambiguity by creating a function of same name in derived class and using the following syntax under it:

   Base_class :: greet();

   where Base_class is that base class of which we want to inherit greet function in derived class.

2. If a class is derived from a single class and a function of derived class is also present in base class then the function of derived class will override the base class's function.

Virtual Base Class



→ A variable x in class A is inherited to class B and C and the same variable x will be inherited to class D twice. This is ambiguous.

→ To avoid this – classes B and C are defined by A as an virtual base class. This can be achieved as follows:

```
class B :virtual public A{
   ......................
};

class C :virtual public A{
   ......................
};
```

Now the class D will get only one copy of x.

Constructors in Derived Class

- We can use constructors in derived classes in C++
- If base class constructor does not have any arguments, there is no need of any constructor in derived class.
- But if there are one or more arguments in the base class constructor, derived class need to pass arguments to the base class constructor.
- If both base and derived classes have constructors, base class constructor is executed first whether there are arguments or not.

Constructors in Multiple Inheritance: In multiple inheritance, base classes constructors are executed in the order in which they appear in the class declaration.

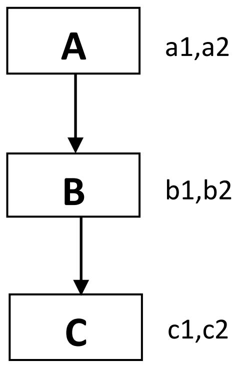
Constructors in Multilevel Inheritance: In multilevel inheritance, the constructors are executed in the order of inheritance.

SPECIAL SYNTAX

- C++ supports a special syntax for passing arguments to multiple base classes.
- The constructor of the derived class receives all the arguments at once and then will pass the calls to the respective base classes.
- The body is called after all the constructors are finished executing.

Syntax for constructor of class C:

```
C (a1, a2, b1, b2, c1, c2): B (b1, b2), A (a1, a2){
    ---body of C---
}
```



Special Case of Virtual Base Class

→The constructors for virtual base classes are invoked before an non-virtual base class.

→ Virtual base class is like VIP.

→If there are multiple virtual base classes, they are invoked in the order declared.

→ Any non-virtual base class are then constructed before the derived class constructor is executed.

E.g.

```
Case1:
class B: public A{
   // Order of execution of constructor -> first
       A() then B()
};

Case2:
class A: public B, public C{
    // Order of execution of constructor -> B()
        then C() and A()
};

Case3:
class A: public B, virtual public C{
    // Order of execution of constructor -> C()
        then B() and A()
};
```