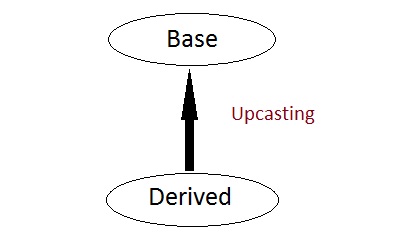
**Upcasting in C++**

Upcasting is using the Super class's reference or pointer to refer to a Sub class's object. Or we can say that, the act of converting a Sub class's reference or pointer into its Super class's reference or pointer is called Upcasting.



class Super

{ int x;

public:

void funBase() { cout << "Super function"; }

};

class Sub : public Super

{ int y;

};

int main()

{

Super\* ptr; // Super class pointer

Sub obj;

ptr = &obj;

Super &ref; // Super class's reference

ref=obj;

}

The opposite of Upcasting is **Downcasting**, in which we convert Super class's reference or pointer into derived class's reference or pointer. We will study more about Downcasting later

**Functions that are never Inherited**

* Constructors and Destructors are never inherited and hence never overrided.
* Also, assignment operator = is never inherited. It can be overloaded but can't be inherited by sub class.

**Inheritance and Static Functions**

1. They are inherited into the derived class.
2. If you redefine a static member function in derived class, all the other overloaded functions in base class are hidden.
3. Static Member functions can never be virtual. We will study about Virtual in coming topics.

**Hybrid Inheritance and Virtual Class**

In Multiple Inheritance, the derived class inherits from more than one base class. Hence, in Multiple Inheritance there are a lot chances of ambiguity.

class A

{ void show(); };

class B:public A {};

class C:public A {};

class D:public B, public C {};

int main()

{

D obj;

obj.show();

}

In this case both class B and C inherits function show() from class A. Hence class D has two inherited copies of function show(). In main() function when we call function show(), then ambiguity arises, because compiler doesn't know which show() function to call. Hence we use **Virtual** keyword while inheriting class.

class B : virtual public A {};

class C : virtual public A {};

class D : public B, public C {};

Now by adding virtual keyword, we tell compiler to call any one out of the two show() funtions.

**Hybrid Inheritance and Constructor call**

As we all know that whenever a derived class object is instantiated, the base class constructor is always called. But in case of Hybrid Inheritance, as discussed in above example, if we create an instance of class D, then following constructors will be called :

* before class D's constructor, constructors of its super classes will be called, hence constructors of class B, class C and class A will be called.
* when constructors of class B and class C are called, they will again make a call to their super class's constructor.

This will result in multiple calls to the constructor of class A, which is undesirable. As there is a single instance of virtual base class which is shared by multiple classes that inherit from it, hence the constructor of the base class is only called once by the constructor of concrete class, which in our case is class D.

If there is any call for initializing the constructor of class A in class B or class C, while creating object of class D, all such calls will be skipped.