**Virtual Destructors**

Destructors in the Base class can be Virtual. Whenever Upcasting is done, Destructors of the Base class must be made virtual for proper destrucstion of the object when the program exits.

**NOTE :** Constructors are never Virtual, only Destructors can be Virtual.

**Upcasting without Virtual Destructor**

Lets first see what happens when we do not have a virtual Base class destructor.

class Base

{

public:

~Base() {cout << "Base Destructor\t"; }

};

class Derived:public Base

{

public:

~Derived() { cout<< "Derived Destructor"; }

};

int main()

{

Base\* b = new Derived; //Upcasting

delete b;

}

Output : Base Destructor

In the above example, **delete b** will only call the Base class destructor, which is undesirable because, then the object of Derived class remains undestructed, because its destructor is never called. Which results in memory leak.

**Upcasting with Virtual Destructor**

Now lets see. what happens when we have Virtual destructor in the base class.

class Base

{

public:

**virtual** ~Base() {cout << "Base Destructor\t"; }

};

class Derived:public Base

{

public:

~Derived() { cout<< "Derived Destructor"; }

};

int main()

{

Base\* b = new Derived; //Upcasting

delete b;

}

**Output :**

Derived Destructor

Base Destructor

When we have Virtual destructor inside the base class, then first Derived class's destructor is called and then Base class's destructor is called, which is the desired behaviour.

**Pure Virtual Destructors**

* Pure Virtual Destructors are legal in C++. Also, pure virtual Destructors must be defined, which is against the pure virtual behaviour.
* The only difference between Virtual and Pure Virtual Destructor is, that pure virtual destructor will make its Base class Abstract, hence you cannot create object of that class.
* There is no requirement of implementing pure virtual destructors in the derived classes.

class Base

{

public:

**virtual** ~Base() = 0; //Pure Virtual Destructor

};

**Base::~Base()** { cout << "Base Destructor"; } //Definition of Pure Virtual Destructor

class Derived:public Base

{

public:

~Derived() { cout<< "Derived Destructor"; }

};