Section 21 : Destructor and Virtual Destructors

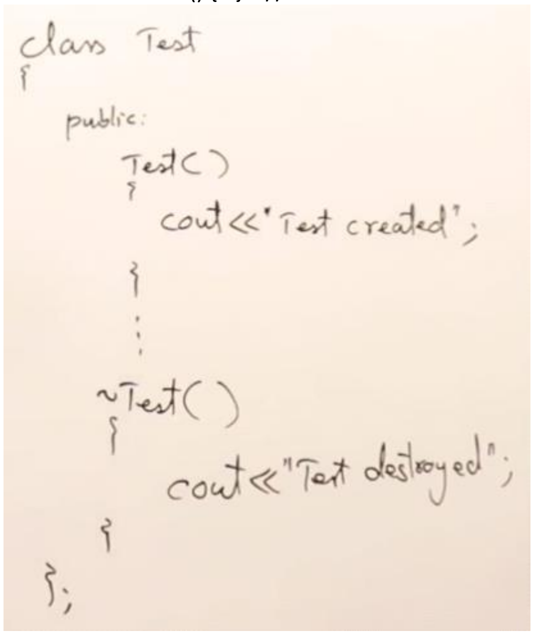
15 April 2022 16:49

© Rajat Kumar
https://www.linkedin.com/in/imRajat/
https://github.com/im-Rajat

Section 21 : Destructor and Virtual Destructors

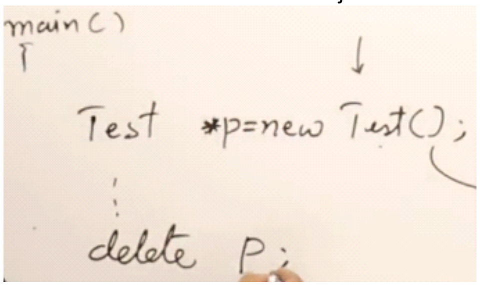
Destructor :

- We know whenever we create an object of class, a constructor will be called.
- We can write a similar thing like constructor, only difference is before the function name tild ~ is used.
- Constructor : Test() {…}
- Destructor : ~Test() {…}   // this function is called when the object is destroyed.



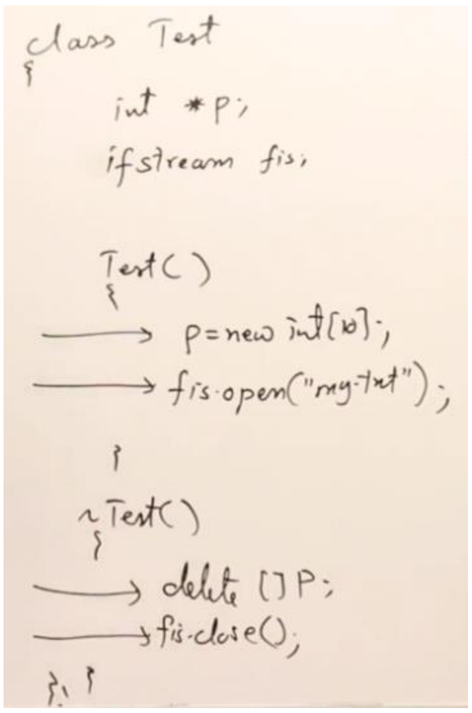
```
class Test
{
   public:
      Test()
      {
         cout<<'Test created';
      }
      :
     ~Test()
     {
         cout<<"Test destroyed";
     }
};
```

- So constructor is called when object is created. The destructor is called when object is destroyed.



```
main()
{
    Test *p=new Test();
    :
    delete p;
```

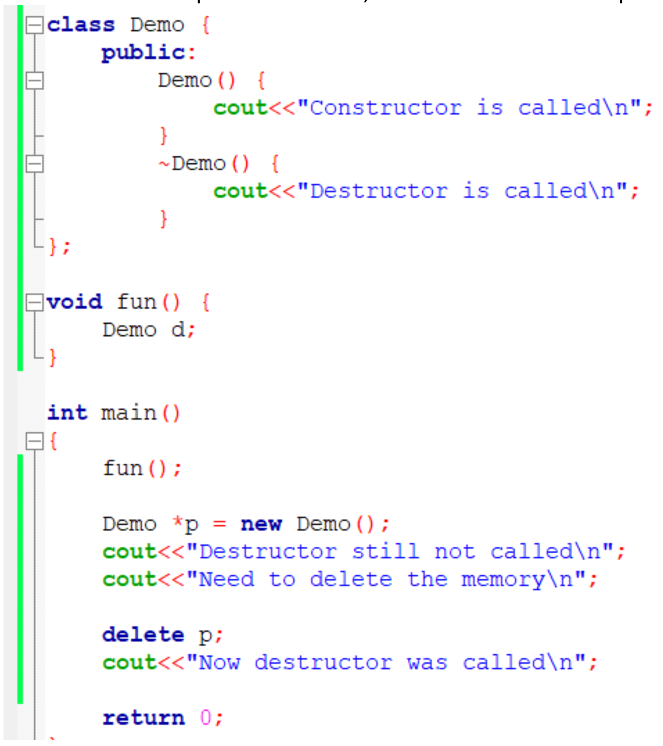
- When we call delete p, it means destructor is called.
- Constructor is used for initialization purposes. It is also used for allocating resources.
- What is the use of destructor? It is used for deallocating resources, releasing the resources.



```
class Test
{
    int *p;
    ifstream fis;

    Test()
    {
  ——→ p=new int[10];
  ——→ fis.open("my.txt");
    }
   ~Test()
    {
  ——→ delete []P;
  ——→ fis.close();
 };
}
```

- Above example where we can see constructor is used for acquiring resources and destructor is used for releasing resources.
- We can have multiple constructors, but we can't have multiple destructors.



```
class Demo {
    public:
        Demo() {
            cout<<"Constructor is called\n";
        }
        ~Demo() {
            cout<<"Destructor is called\n";
        }
};

void fun() {
    Demo d;
}

int main()
{
    fun();

    Demo *p = new Demo();
    cout<<"Destructor still not called\n";
    cout<<"Need to delete the memory\n";

    delete p;
    cout<<"Now destructor was called\n";

    return 0;
```

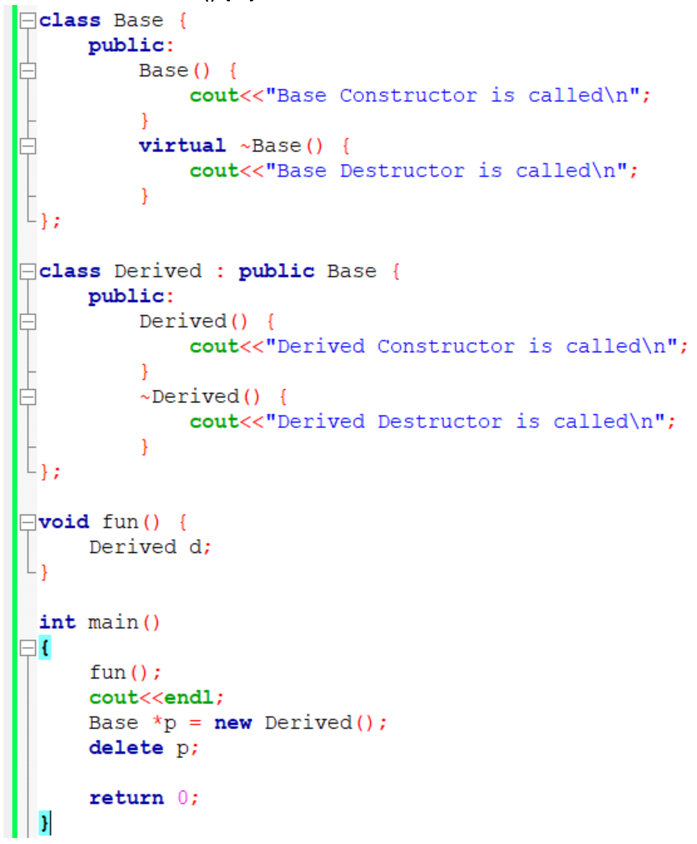
Destructor in Inheritance :

- How constructor and destructor are called when we create an object of Derived class :
  - Derived d;
  - Calling of Constructor is as follows : (Top to Bottom)
    - Base Constructor
    - Derived Constructor
  - Calling of destructor is as follows : (Bottom to Top)
    - Derived Destructor
    - Base Destructor

Virtual Destructor :

- Base *p = new Derived(); …….. delete p;
- In C++ the functions are called depending on the pointer, not upon the object.
- Pointer is of base class, so only base class destructor will be called.
- C++ compiler thinks that the object is of base class as we are using base pointer.
- So when we call delete p only Base Destructor will called.

- But we want to work it as normal, first destructor of derived then base.
- To work we have to write down virtual before base class destructor.
    - Virtual ~Base() {…}



```
class Base {
    public:
        Base() {
            cout<<"Base Constructor is called\n";
        }
        virtual ~Base() {
            cout<<"Base Destructor is called\n";
        }
};

class Derived : public Base {
    public:
        Derived() {
            cout<<"Derived Constructor is called\n";
        }
        ~Derived() {
            cout<<"Derived Destructor is called\n";
        }
};

void fun() {
    Derived d;
}

int main()
{
    fun();
    cout<<endl;
    Base *p = new Derived();
    delete p;

    return 0;
}
```

- It is useful for runtime polymorphism.

- Destructor : A special function that is called to free the resources, acquired by the object.
- When a destructor is called : Just before the end of object life.
- If in multiple inheritance, class C inherits class B, and Class B inherits class A. In which sequence are their destructors called, if an object of class C was destroyed?
    - ~C() then ~B() then ~A()
- When is the destructor of a global object called?
    - Just before end of program.