Destructor

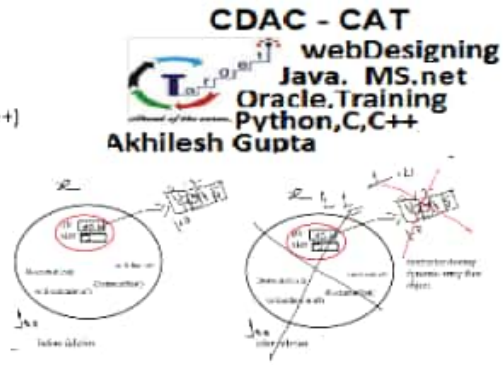
- If you want to perform any additional task while of the destroying of object such as clean up, operations garbage collection, break the connection etc.
- *It also has the same name as class name* **with prefix symbol ~ (Tilde)**

```
class Demo{
        ~Demo(){  // destructor
        }
};
```

- They do not have return value declaration and not even void.
- Destructor does not have parameter so we cannot overload the destructor.
- You can't call destructor the way you call a normal function.
- It automatically calls destructor when destroying objects.
- Destructor can be ***virtual*** or but it cannot be static.

e.g.

```
#include<iostream.h>
class DestructorTest
{
        private:
                int *ptr ;
                int size;
        public:
                DestructorTest()
                {
                        ptr = new int[5] ;
                        size = 5 ;

                }
                void setdata(int a[])
                {
                        for ( int i = 0 ; i < size ; i ++)
                        {
                                ptr[i] = a[i] ;
                        }
                }
                void display()
                {
                        for ( int i = 0 ; i < size ; i ++)
                        {
                                cout << ptr[i];
                        }
                }
                ~DestructorTest()
                {
                        delete []ptr ;
                }

};
main()
```



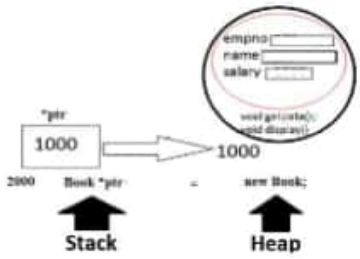
```
{
        DestructorTest x;
        int a[]={1,2,3,4,5};
        x.setdata(a);
        x.display();
        delay(20000);

}
```

```
using namepsace std ;
#define size 5
main()
{
        int a[size];
        int i, sum ;
        float avg ;
        cout << "Enter " << size << "values";

        for( i = 0 ; i< size ; i++)
        {
                cin >> a[i];
        }
        sum = 0 ;
        for( i = 0 ; i< size ; i++)
        {
                sum = sum + a[i];
        }
        avg = (float)sum / size;
        cout << endl <<"average is " << avg ;
}
```

```
#include<iostream>
using namespace std ;
#define size 5
main()
{
        int i, sum ;
        float avg ;
        cout << "Enter " << size << "values";
        int *a = new int[size];
        for( i = 0 ; i< size ; i++)
        {
                cin >> a[i];
        }
        sum = 0 ;
        for( i = 0 ; i< size ; i++)
        {
                sum = sum + a[i];
        }
        avg = (float)sum / size;
        cout << endl <<"average is " << avg ;
}
```


- ***We can create dynamic object as well.***

```
#include<iostream>
using namespace std;
class book
{
        private:
                char name[20];
                int page;
                float price;
        public:
                void getdata()
                {
                        cout << endl << "Enter book name";
                        cin >> name;
                        cout << endl << "Enter book page';
                        cin >> page;
                        cout << endl << "Enter book price";
                        cin >> price;
                }
                void display()
                {
                        cout << endl << "Book detail #1";
                        cout << endl << "book name : "<< name;
                        cout << endl << "book page : "  << page ;
                        cout << endl << "book price : " << price;

                }
```

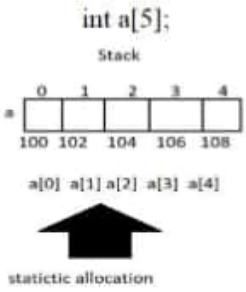



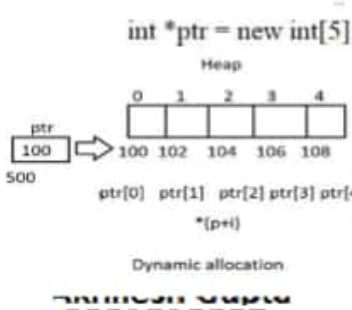
- ***Memory Allocation***
  - Element (***variable, array, structure, string, object etc.)*** of technology can be form ***statically or dynamically.***
  - ***Memory allocation for the elements are two type***
    - ***Statically memory allocation***
      - In such type of allotment the decision against the memory allotment is done at the ***compilation time (at the translation stage)***.
        e.g.

        int a[100] ; //In this case array a have 100 int elements

        - Drawback of static allocation
          - Suppose we use 48 then remaining elements are waste.
          - If need more element then the total size need to change the program

    - ***Dynamically allocation***
      - ***Dynamic memory management refers to manual memory management.***
      - ***This allows you to obtain more memory when required and release it when not necessary.***
      - In such type of allotment the decision against the memory take place at the run time so it is called ***dynamic allocation.***





  - We can allocate element dynamically.
  - ***Operating system is responsible for allotment of the memory.***

```
};

main()
{
        book *ptr = new book;
        ptr -> getdata();
        ptr -> display();
        delete ptr ;
}
```

- ***C++ introduces two operators for dynamic memory management they are as follow***
  - ***new***
    - ***To create dynamic object***
  - ***delete***
    - ***to destroy dynamic object***
- They place at the category number ***two in precedence table.***
- new allocate the dynamic object in ***heap memory***
- It is implicitly called by constructor.
- An object create by ***new exist until it explicitly destroyed by the delete operator.***
- Dynamic objects are not local and global they are ***reachable or unreachable***.

Note:
delete ptr ;

- Delete ptr does not means to delete the ptr, it means to delete the object which is pointed by the ptr ;