Destructor

Destructor:-

A destructor is a special member function that works just opposite to a constructor; unlike constructors that are used for initializing an object, destructors destroy (or delete) the object. The purpose of the destructor is to free the resources that the object may have acquired during its lifetime.

```
~class_name()
{
   //Some code
}
```

Like the constructor, the destructor name should exactly match the class name. A destructor declaration should always begin with the tilde(~) symbol, as shown in the syntax above.

The thing is to be noted here, if the object is created by using new or the constructor uses new to allocate memory that resides in the heap memory or the free store, the destructor should use delete to free the memory.

Example:-

```
#include <iostream>
using namespace std;
class Guided_path{
    public:
    //Constructor
    Guided_path ()
    {
        cout << "Constructor is called" << endl;
        cout<<"Welcome to Guided Path"<< endl;
    }
    //Destructor
    ~Guided_path ()
    {
```

```
        cout<< "Happy Learning"<< endl;
        cout << "Destructor is called" << endl;
    }
};

int main ()
{
  //Object created
  Guided_path obj;
  // at the end object destructed
}

Output:
Constructor is called
Welcome to Guided Path
Happy Learning
Destructor is called
```

When is a destructor called?

A destructor function is called automatically when:

- ➔ the object goes out of scope
- ➔ the program ends
- ➔ a scope (the { } parenthesis) containing local variable ends.
- ➔ a delete operator is called

Destructor rules

- ❖ The name should begin with a tilde sign(~) and match the class name.
- ❖ There cannot be more than one destructor in a class.
- ❖ Unlike constructors that can have parameters, destructors do not allow any parameter.
- ❖ They do not have any return type, not even void. I
- ❖ A destructor should be declared in the public section of the class.
- ❖ The programmer cannot access the address of the destructor.
- ❖ It has no return type, not even void.
- ❖ When you do not specify any destructor in a class, the compiler generates a default destructor and inserts it into your code.