What is a destructor?

Destructor is an instance member function which is invoked automatically whenever an object is going to be destroyed. Meaning, a destructor is the last function that is going to be called before an object is destroyed.

Properties of C++ Destructor

1. When objects are destroyed, the destructor function is automatically named.
2. It's not possible to declare it static or const.
3. There are no arguments for the destructor.
4. It doesn't even have a void return form.
5. A member of the union cannot be an entity of a class with a destructor.
6. In the public section of the class, a destructor should be declared.
7. The destructor's address is inaccessible to the programmer.

When is a Destructor Called?

The destructor will be called automatically when an object is no longer in scope and also in situations mentioned below

(1) The operation is completed

(2) The program is completed

(3) A block containing local variables is completed

(4) The delete operator is used

To identify the destructor from the normal function

1. Destructors have the same name as the class, but with a tilde () before it.
2. Destructors don't take arguments and don't give them back.

Rules for C++ Destructors

According to C++ destructor the statement should begin with a tilde () and the same class name.

1. Destructors are not equipped with parameters or a return form.
2. Destructors are invoked automatically and cannot be invoked manually from a program.
3. Destructors cannot be overloaded.

4. Which can be a virtual destructor.
5. The Destructor will execute the reverse order of object creation.
6. Destructor always present only in the public section.

Use of C++ Destructor

- Object memory getting released.
- The pointer variables' memory getting released.
- Files and services getting closed.

Syntax:

Syntax for defining the destructor within the class

```
~ <class-name>()
{

}
```

Syntax for defining the destructor outside the class

```
<class-name>: : ~ <class-name>()
{

}
```

```
// Example:

#include<iostream>
using namespace std;

class Test
{
public:
            Test()
   {
                        cout<<"\n Constructor executed";
   }

   ~Test()
            {
                        cout<<"\n Destructor executed";
            }
};
main()
{
   Test t;
```

```
    return 0;
}
```

Output

```
Constructor executed
 Destructor executed
```

```
// Example:
#include<iostream>
using namespace std;
class Test
{
    public:
            Test()
            {
                    cout<<"\n Constructor executed";
            }

            ~Test()
            {
                    cout<<"\n Destructor executed";
            }
};

main()
{
    Test t,t1,t2,t3;
    return 0;
}
```

Output

```
Constructor executed
 Constructor executed
 Constructor executed
 Constructor executed
 Destructor executed
 Destructor executed
 Destructor executed
 Destructor executed
```

```
// Example:
#include<iostream>
using namespace std;
int count=0;
class Test
{
   public:
          Test()
          {
                 count++;
                 cout<<"\n No. of Object created:\t"<<count;
          }

          ~Test()
          {
                 cout<<"\n No. of Object destroyed:\t"<<count;
                 --count;
          }
};

main()
{
   Test t,t1,t2,t3;
   return 0;
}
```

Output

```
No. of Object created:    1
 No. of Object created:    2
 No. of Object created:    3
 No. of Object created:    4
 No. of Object destroyed:    4
 No. of Object destroyed:    3
 No. of Object destroyed:    2
 No. of Object destroyed:    1
```

Properties of Destructor:

- Destructor function is automatically invoked when the objects are destroyed.
- It cannot be declared static or const.
- The destructor does not have arguments.
- It has no return type not even void.
- An object of a class with a Destructor cannot become a member of the union.
- A destructor should be declared in the public section of the class.

- The programmer cannot access the address of destructor.

When is destructor called?

A destructor function is called automatically when the object goes out of scope:

(1) the function ends

(2) the program ends

(3) a block containing local variables ends

(4) a delete operator is called

Note: **destructor** can also be called explicitly for an object.
syntax:
object_name.~class_name()

How are destructors different from a normal member function?

Destructors have same name as the class preceded by a tilde (~)

Destructors don't take any argument and don't return anything

Can there be more than one destructor in a class?

No, there can only one destructor in a class with classname preceded by ~, no parameters and no return type.

When do we need to write a user-defined destructor?

If we do not write our own destructor in class, compiler creates a default destructor for us. The default destructor works fine unless we have dynamically allocated memory or pointer in class. When a class contains a pointer to memory allocated in class, we should write a destructor to release memory before the class instance is destroyed. This must be done to avoid memory leak.