Destructors in C++

Difficulty Level : Easy • Last Updated : 11 Dec, 2022

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.

- Destructor is also a special member function like constructor. Destructor destroys the class objects created by constructor.
- Destructor has the same name as their class name preceded by a tilde (~) symbol.
- It is not possible to define more than one destructor.
- The destructor is only one way to destroy the object create by constructor. Hence destructor can-not be overloaded.
- Destructor neither requires any argument nor returns any value.
- It is automatically called when object goes out of scope.
- Destructor release memory space occupied by the objects created by constructor.
- In destructor, objects are destroyed in the reverse of an object creation.

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

Syntax:

```
Syntax for defining the destructor within the class
~ <class-name>()
{
```

```
}
```

Syntax for defining the destructor outside the class

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

}
```

C++

```
// 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
```

C++

```
// 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
```

C++

```
// 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

---

CPP

```
class String {
private:
    char* s;
    int size;

public:
    String(char*); // constructor
    ~String(); // destructor
};

String::String(char* c)
{
    size = strlen(c);
    s = new char[size + 1];
    strcpy(s, c);
}
String::~String() { delete[] s; }
```

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.

Can a destructor be virtual?

Yes, In fact, it is always a good idea to make destructors virtual in base class when we have

a virtual function. See virtual destructor for more details.

You may like to take a quiz on destructors.

Related Articles :

Constructors in C++

Virtual Destructor

Pure virtual destructor in C++

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

263

Related Articles

1. Playing with Destructors in C++
2. C++ Interview questions based on constructors/ Destructors.
3. C++ Error - Does not name a type
4. Execution Policy of STL Algorithms in Modern C++
5. Square Meter (Meter Squared)
6. C++ Program To Print Pyramid Patterns
7. Jagged Arrays in C++
8. Introduction to Parallel Programming with OpenMP in C++
9. Hollow Half Pyramid Pattern Using Numbers
10. 30 OOPs Interview Questions and Answers (2023)

Previous Next

Article Contributed By :

GeeksforGeeks