Constructors in C++

Constructor is a special member function of a class that initializes the object of the class.

The Compiler calls the Constructor whenever an object is created. Constructors initialize values to object members after storage is allocated to the object.

There can be two types of constructors in C++.

1. Default constructor
2. Parameterized constructor
3. Copy Constructor

Note: If you do not specify a constructor, the compiler generates a default constructor for you (expects no parameters and has an empty body).

In C++, a constructor has the same name as that of the class and it does not have a return type. For example,

```
class Student {
   public:
    // create a constructor
    Student() {
        // Object Initialization
    }
};
```

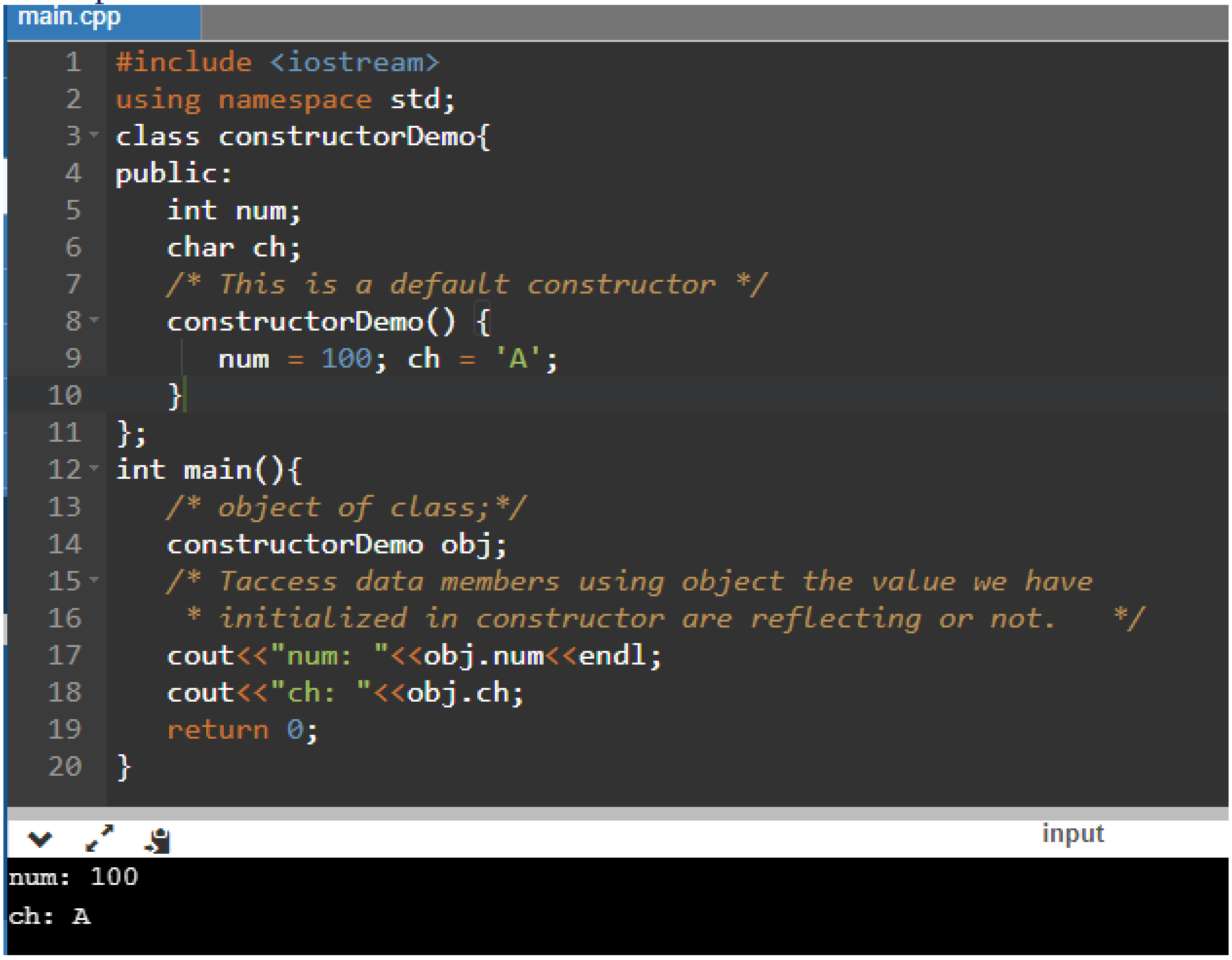
Here, the function `Student ()` is a constructor of the class `Student`. Notice that the constructor

1. has the same name as the class
2. does not have a return type, and
3. is `public`

Default Constructor

A constructor with no parameters is known as a **default constructor**. In the example above, `Student ()` is a default constructor.

Example 1: C++ Default Constructor



main.cpp

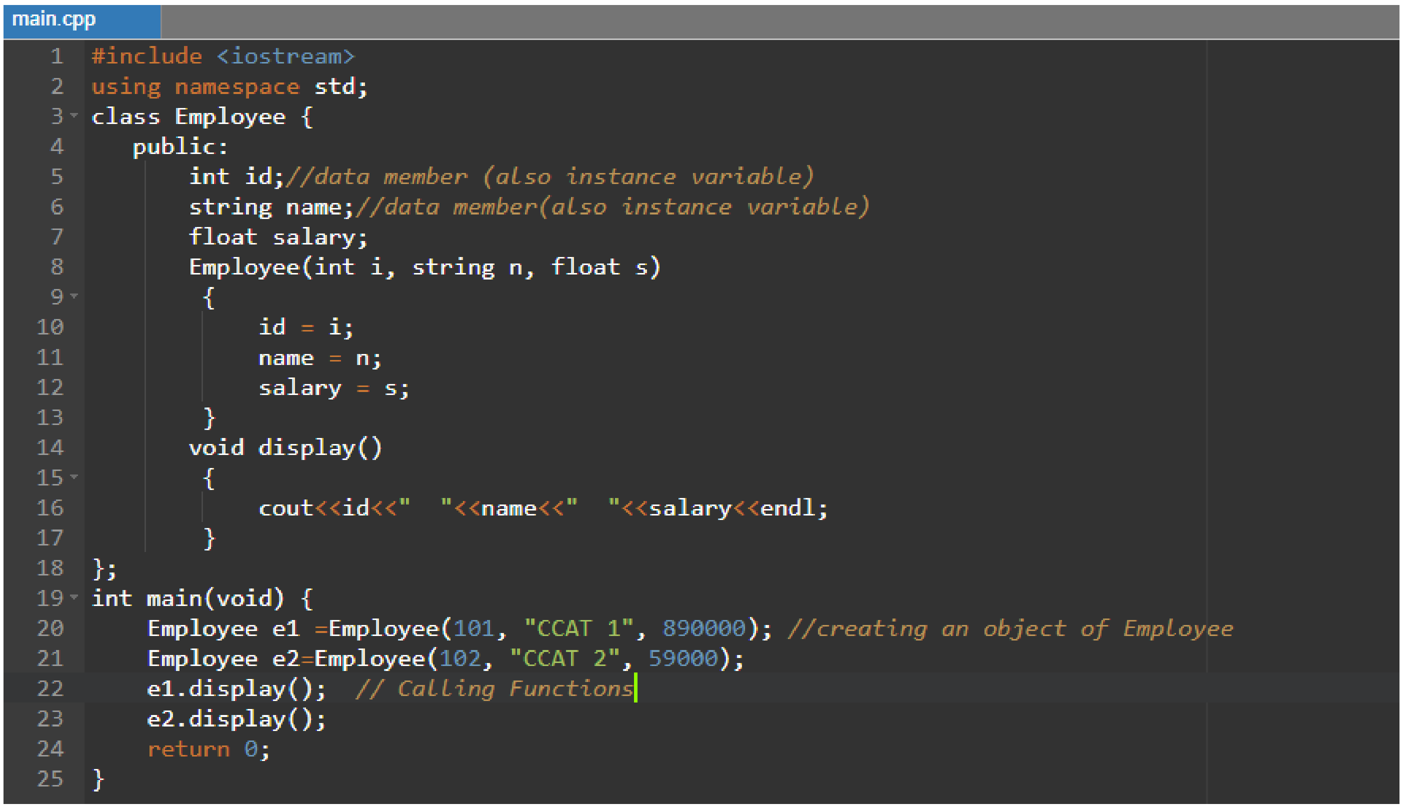
```cpp
#include <iostream>
using namespace std;
class constructorDemo{
public:
    int num;
    char ch;
    /* This is a default constructor */
    constructorDemo() {
        num = 100; ch = 'A';
    }
};
int main(){
    /* object of class;*/
    constructorDemo obj;
    /* Taccess data members using object the value we have
     * initialized in constructor are reflecting or not.    */
    cout<<"num: "<<obj.num<<endl;
    cout<<"ch: "<<obj.ch;
    return 0;
}
```

input

```
num: 100
ch: A
```

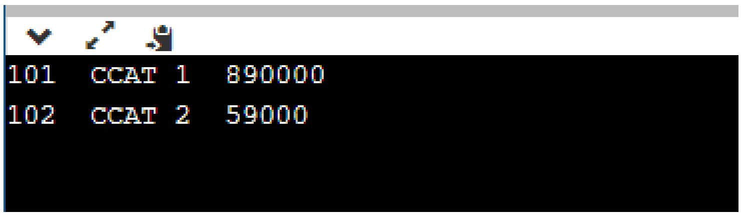
Parameterized Constructor

A constructor which has parameters is called parameterized constructor. Using this Constructor, you can provide different values to data members of different objects, by passing the appropriate values as argument.



main.cpp

```cpp
#include <iostream>
using namespace std;
class Employee {
   public:
       int id;//data member (also instance variable)
       string name;//data member(also instance variable)
       float salary;
       Employee(int i, string n, float s)
        {
             id = i;
             name = n;
             salary = s;
        }
       void display()
        {
             cout<<id<<"  "<<name<<"  "<<salary<<endl;
        }
};
int main(void) {
    Employee e1 =Employee(101, "CCAT 1", 890000); //creating an object of Employee
    Employee e2=Employee(102, "CCAT 2", 59000);
    e1.display();  // Calling Functions
    e2.display();
    return 0;
}
```



```
101   CCAT 1   890000
102   CCAT 2   59000
```

By using parameterized constructor in above case, we have initialized 2 objects with user defined values. We can have any number of parameters in a constructor.

Destructor

A destructor works opposite to constructor; it destructs the objects of classes. It can be defined only once in a class. Like constructors, it is invoked automatically.by the compiler when the object goes out of scope.

A destructor is defined like constructor. It must have same name as class. But it is prefixed with a tilde sign (~).

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.

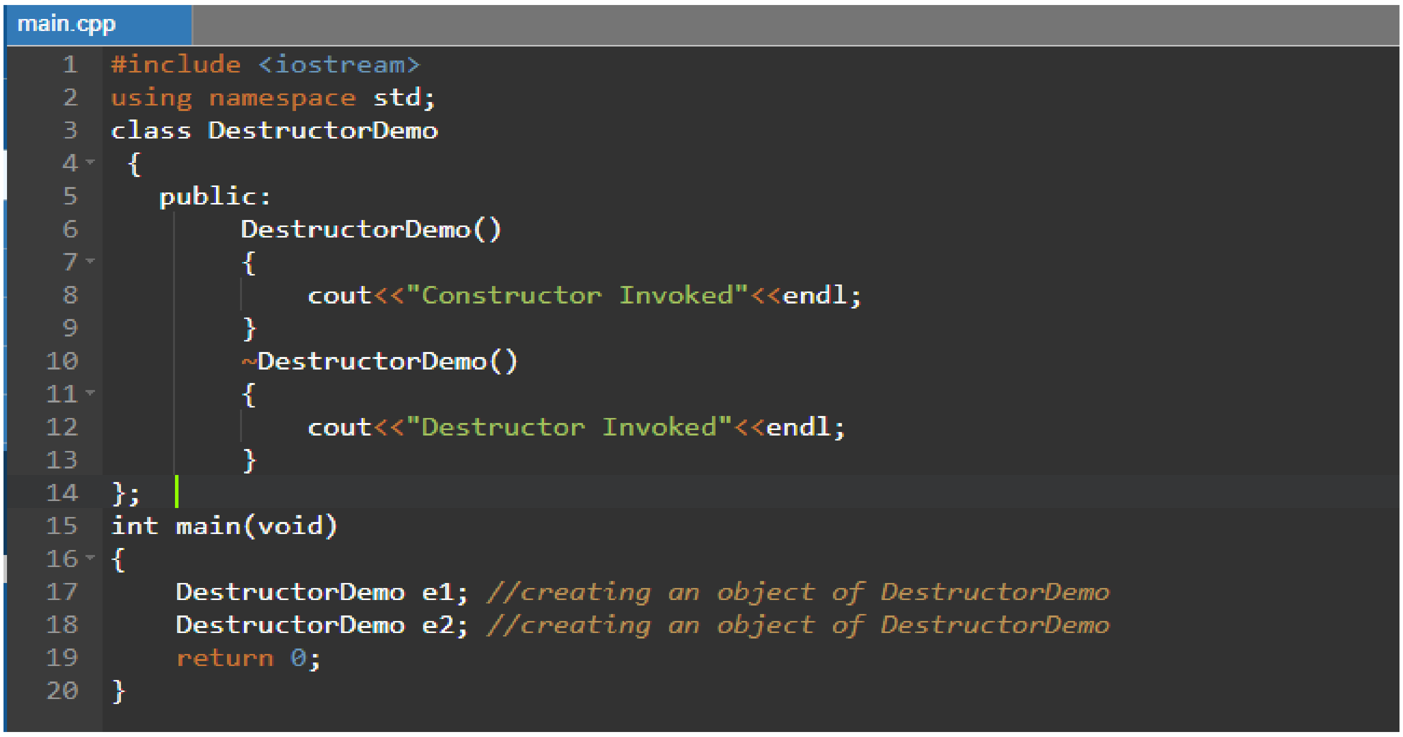
When is destructor called?

(1) the function ends

(2) the program ends

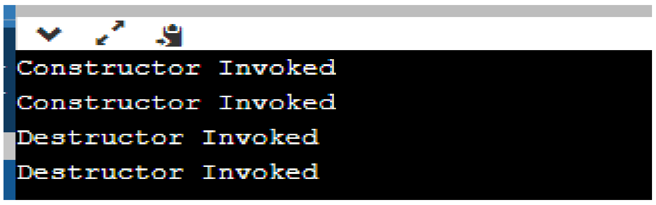
(3) a block containing local variables ends

(4) a delete operator is called



main.cpp

```cpp
#include <iostream>
using namespace std;
class DestructorDemo
 {
   public:
        DestructorDemo()
        {
            cout<<"Constructor Invoked"<<endl;
        }
        ~DestructorDemo()
        {
            cout<<"Destructor Invoked"<<endl;
        }
};
int main(void)
{
    DestructorDemo e1; //creating an object of DestructorDemo
    DestructorDemo e2; //creating an object of DestructorDemo
    return 0;
}
```

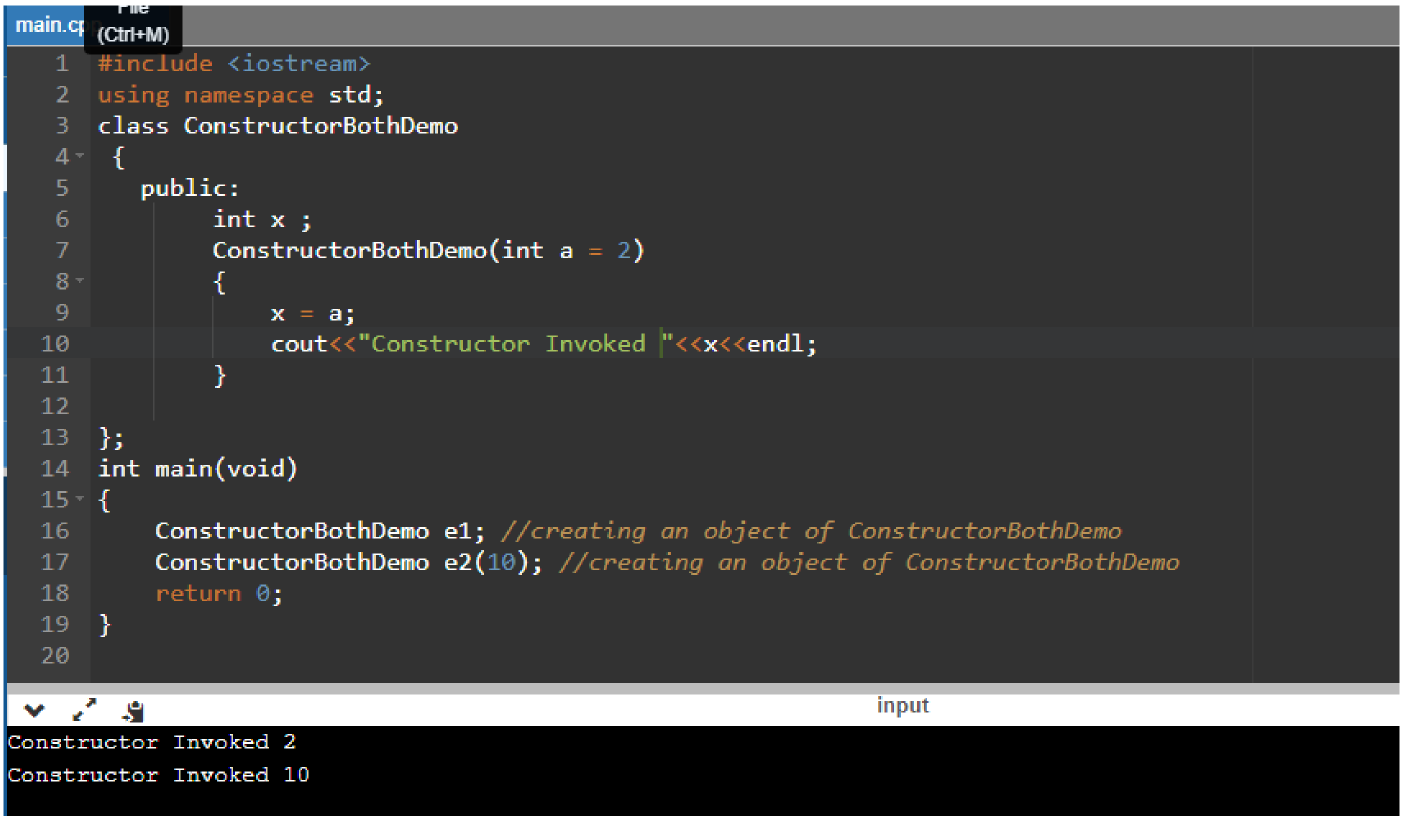


```
Constructor Invoked
Constructor Invoked
Destructor Invoked
Destructor Invoked
```

Join Our Telegram Group to Get Notifications, Study Materials, Practice test & quiz:
https://t.me/ccatpreparations Visit: https://ccatpreparation.com

Single Definition for both Default and Parameterized Constructor

In this example we will use **default argument** to have a single definition for both default and parameterized constructor.



```cpp
#include <iostream>
using namespace std;
class ConstructorBothDemo
 {
   public:
        int x ;
        ConstructorBothDemo(int a = 2)
        {
            x = a;
            cout<<"Constructor Invoked "<<x<<endl;
        }

};
int main(void)
{
    ConstructorBothDemo e1; //creating an object of ConstructorBothDemo
    ConstructorBothDemo e2(10); //creating an object of ConstructorBothDemo
    return 0;
}

```

```
Constructor Invoked 2
Constructor Invoked 10
```