Private Destructor in C++

Difficulty Level : Medium • Last Updated : 17 Jan, 2023

Destructors with the access modifier as private are known as Private Destructors. Whenever we want to prevent the destruction of an object, we can make the destructor private.

What is the use of private destructor?

Whenever we want to control the destruction of objects of a class, we make the destructor private. For dynamically created objects, it may happen that you pass a pointer to the object to a function and the function deletes the object. If the object is referred after the function call, the reference will become dangling.

Predict the Output of the Following Programs:

CPP

```
// CPP program to illustrate
// Private Destructor
#include <iostream>
using namespace std;

class Test {
private:
```

```
    ~Test() {}
};
int main() {}
```

The above program compiles and runs fine. Hence, we can say that: It is **not** a compiler error to create private destructors.
Now, What do you say about the below program?

CPP

```
// CPP program to illustrate
// Private Destructor
#include <iostream>
using namespace std;

class Test {
private:
    ~Test() {}
};
int main() { Test t; }
```

Output

```
prog.cpp: In function ‘int main()’:
prog.cpp:8:5: error: ‘Test::~Test()’ is private
    ~Test() {}
    ^
prog.cpp:10:19: error: within this context
int main() { Test t; }
```

The above program fails in the compilation. The compiler notices that the local variable ‘t’ cannot be destructed because the destructor is private.
Now, What about the Below Program?

CPP

```
// CPP program to illustrate
// Private Destructor
#include <iostream>
using namespace std;

class Test {
private:
    ~Test() {}
};
int main() { Test* t; }
```

The above program works fine. There is no object being constructed, the program just creates a pointer of type "Test *", so nothing is destructed.
Next, What about the below program?

CPP

```cpp
// CPP program to illustrate
// Private Destructor
#include <iostream>
using namespace std;

class Test {
private:
    ~Test() {}
};
int main() { Test* t = new Test; }
```

The above program also works fine. When something is created using dynamic memory allocation, it is the programmer's responsibility to delete it. So compiler doesn't bother.
In the case where the destructor is declared private, an instance of the class can also be created using the malloc() function. The same is implemented in the below program.

CPP

```cpp
// CPP program to illustrate
// Private Destructor

#include <bits/stdc++.h>
using namespace std;

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

private:
    ~Test() // Private Destructor
    {
        cout << "Destructor called\n";
    }
};

int main()
{
    Test* t = (Test*)malloc(sizeof(Test));
    return 0;
```

```
}
```

The above program also works fine. However, The below program fails in the compilation. When we call delete, destructor is called.

CPP

```cpp
// CPP program to illustrate
// Private Destructor
#include <iostream>
using namespace std;

class Test {
private:
    ~Test() {}
};

// Driver Code
int main()
{
    Test* t = new Test;
    delete t;
}
```

We noticed in the above programs when a class has a private destructor, only dynamic objects of that class can be created. Following is a way to **create classes with private destructors and have a function as a friend of the class.** The function can only delete the objects.

CPP

```cpp
// CPP program to illustrate
// Private Destructor
#include <iostream>

// A class with private destructor
class Test {
private:
    ~Test() {}

public:
    friend void destructTest(Test*);
};

// Only this function can destruct objects of Test
void destructTest(Test* ptr) { delete ptr; }

int main()
{
```

```
    // create an object
    Test* ptr = new Test;

    // destruct the object
    destructTest(ptr);

    return 0;
}
```

Another way to use private destructors is by using **the class instance method**.

C++

```
#include <iostream>

using namespace std;

class parent {
    // private destructor
    ~parent() { cout << "destructor called" << endl; }

public:
    parent() { cout << "constructor called" << endl; }
    void destruct() { delete this; }
};

int main()
{
    parent* p;
    p = new parent;
    // destructor called
    p->destruct();

    return 0;
}
```

Output

```
constructor called
destructor called
```

Must Read: Can a Constructor be Private in C++?

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

156

Related Articles