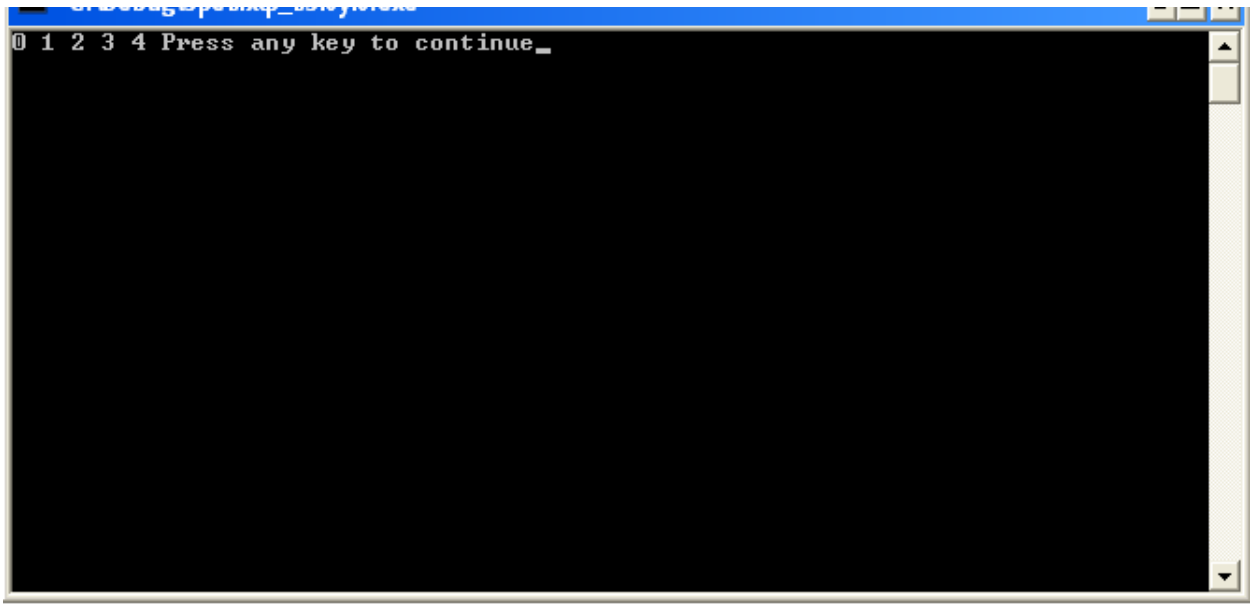
Static

```
#include <iostream>
#include <string>
using namespace std;

void demo()
{
    // static variable
    static int count = 0;
    cout << count << " ";

    // value is updated and
    // will be carried to next
    // function calls
    count++;
}

int main()
{
    for (int i=0; i<5; i++)
        demo();
    return 0;
}
```




```
// C++ program to demonstrate static
// variables inside a class

#include<iostream>
using namespace std;

class Demo
{
   public:
     static int i;

     Demo()
```

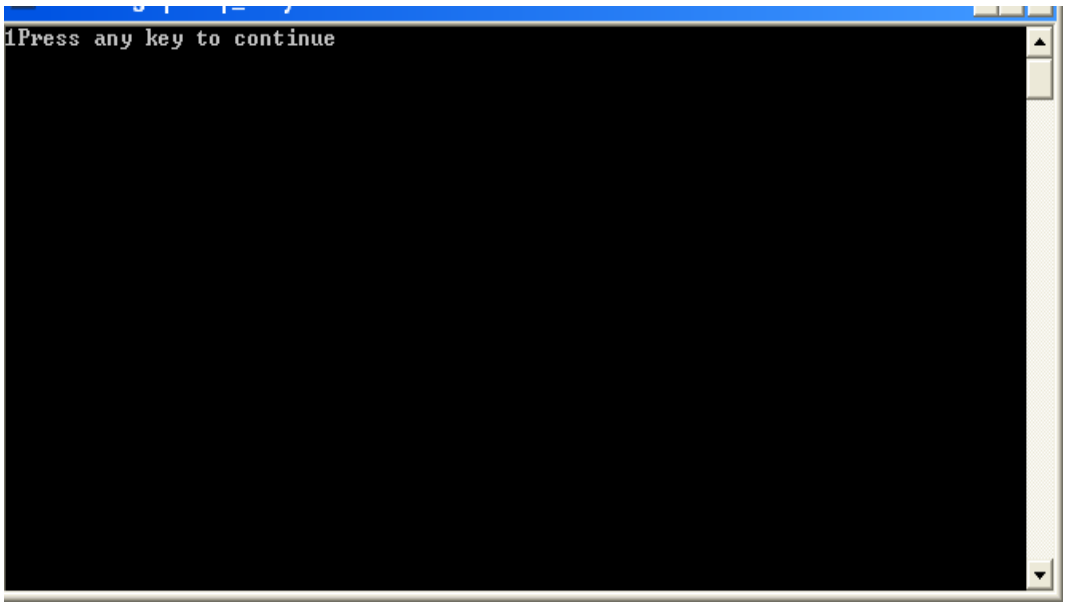
```
    {
       // Do nothing
    };
};

int main()
{
  Demo obj1;
  Demo obj2;
  obj1.i =2;
  obj2.i = 3;

  // prints value of i
  cout << obj1.i<<" "<<obj2.i;
}
```

Получим ошибку
error LNK2001: unresolved external symbol "public: static int Demo::i" (?i@GfG@@2HA)

В приведенной выше программе вы можете видеть, что мы пытались создать несколько копий статической переменной i для нескольких объектов. Но этого не произошло. Таким образом, статическая переменная внутри класса должна быть явно инициализирована пользователем с использованием имени класса и оператора разрешения области видимости вне класса, как показано ниже

```
#include<iostream>
using namespace std;

class Demo
{
public:
    static int i;

    Demo()
    {
        // Do nothing
    };
};

int Demo::i = 1;

int main()
{
    Demo obj;
    // prints value of i
    cout << obj.i;
}
```

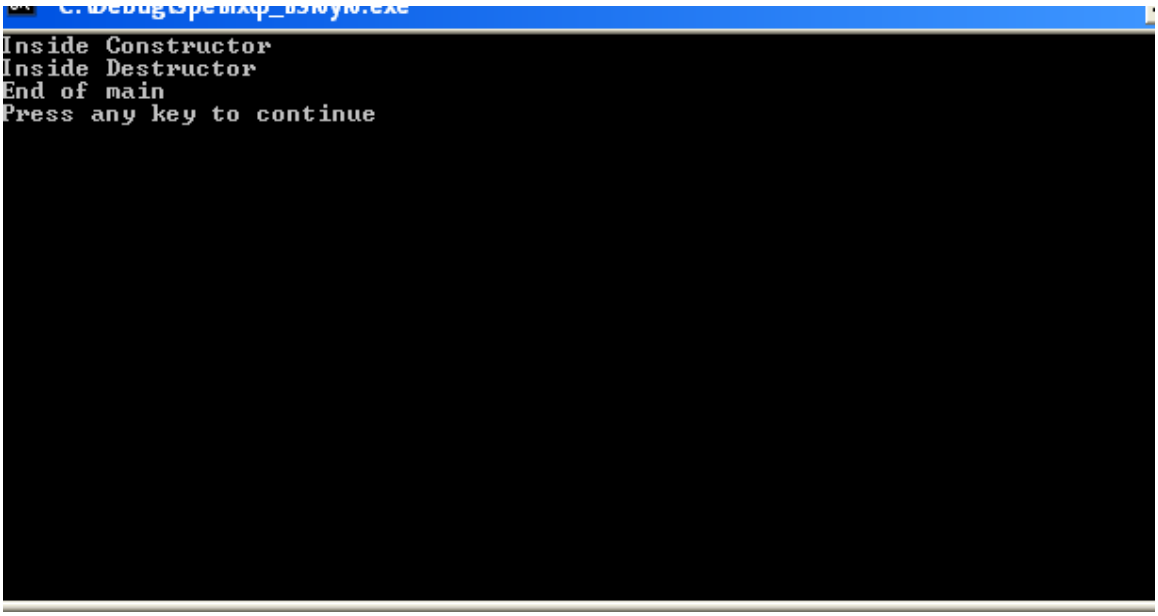
- Output:
- 1

- 

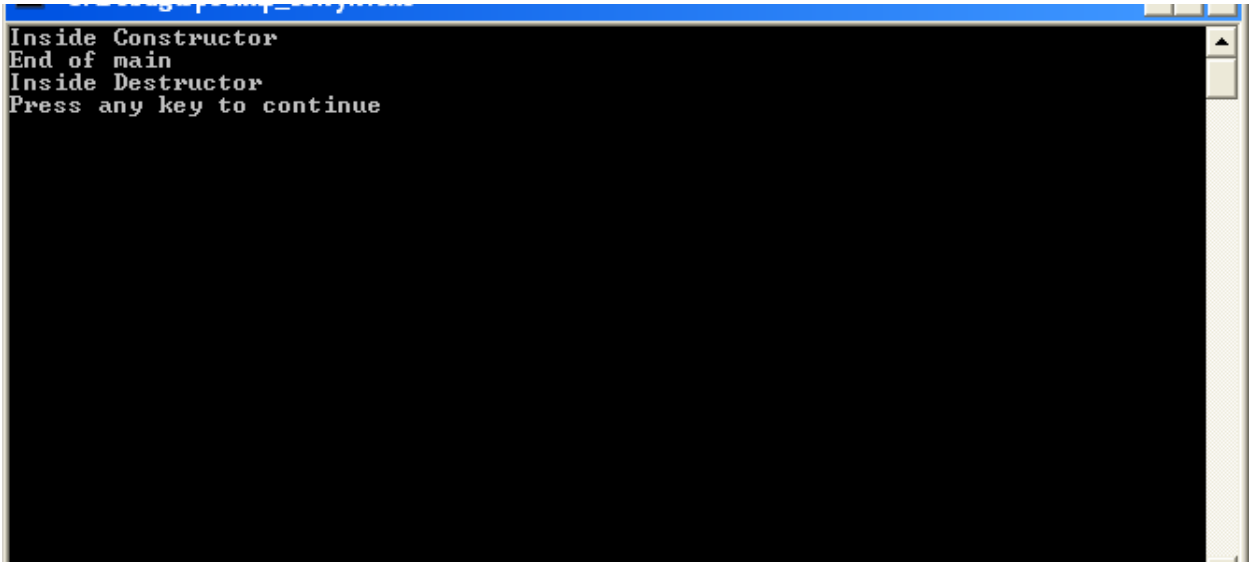
Static Members of Class

- .

```
// CPP program to illustrate
// when not using static keyword
#include<iostream>
using namespace std;

class Demo
{
    int i;
    public:
        Demo ()
        {
            i = 0;
            cout << "Inside Constructor\n";
        }
        ~ Demo ()
        {
            cout << "Inside Destructor\n";
        }
};

int main()
{
    int x = 0;
    if (x==0)
    {
        Demo obj;
    }
    cout << "End of main\n";
}
```




```
#include<iostream>
using namespace std;

class GfG
{

    public:
    Demo()
    {

        cout << "Inside Constructor\n";
    }

    ~Demo ()
    {
        cout << "Inside Destructor\n";
    }
};

int main()
{
    int x = 0;
    if (x==0)
    {
        static Demo obj;
    }
    cout << "End of main\n";
}
```



- Вы можете четко видеть изменение вывода. Теперь деструктор вызывается после окончания main. Это произошло потому что время жизни статического объекта составляет все время жизни программы.