Static Keyword in C++

Static keyword has different meanings when used with different types. We can use static keyword with:

Static Variables : Variables in a function, Variables in a class

Static Members of Class : Class objects and Functions in a class

Let us now look at each one of these use of static in details:

Static Variables

- **Static variables in a Function**: When a variable is declared as static, space for **it gets allocated for the lifetime of the program**. Even if the function is called multiple times, space for the static variable is allocated only once and the value of variable in the previous call gets carried through the next function call. This is useful for implementing coroutines in C/C++ or any other application where previous state of function needs to be stored.

```
// C++ program to demonstrate
// the use of static Static
// variables in a Function
#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;
}
```

Output

`0 1 2 3 4`

You can see in the above program that the variable count is declared as static. So, its value is carried through the function calls. The variable count is not getting initialized for every time the function is called.

- **Static variables in a class**: As the variables declared as static are initialized only once as they are allocated space in separate static storage so, the static variables **in a class are shared by the objects.** There can not be multiple copies of same static variables for different objects. Also because of this reason static variables can not be initialized using constructors.

```
// C++ program to demonstrate static
// variables inside a class
#include<iostream>
using namespace std;

class GfG{
public:
    static int i;

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

int main()
{
GfG obj1;
GfG obj2;
obj1.i =2;
obj2.i = 3;
// prints value of icout << obj1.i<<" "<<obj2.i;
}
```

You can see in the above program that we have tried to create multiple copies of the static variable i for multiple objects. But this didn't happen. So, a static variable inside a class should be initialized explicitly by the user using the class name and scope resolution operator outside the class as shown below:

```
// C++ program to demonstrate static
// variables inside a class
#include<iostream>
using namespace std;

class GfG{
public:
    static int i;
    GfG()
    {
        // Do nothing   };
};

int GfG::i = 1;
int main()
{
    GfG obj;
    // prints value of i    cout << obj.i;
}
```

Output

```
1
```

Static Members of Class

- **Class objects as static**: Just like variables, objects also when declared as static have a scope till the lifetime of program. Consider the below program where the object is non-static.

```
// CPP program to illustrate
// when not using static keyword
```

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

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

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

Output

```
Inside Constructor
Inside Destructor
End of main
```

In the above program the object is declared inside the if block as non-static. So, the scope of variable is inside the if block only. So when the object is created the constructor is invoked and soon as the control of if block gets over the destructor is invoked as the scope of object is inside the if block only where it is declared.

Let us now see the change in output if we declare the object as static.

```
// CPP program to illustrate
// class objects as static
#include<iostream>
using namespace std;

class GfG{
    int i = 0;

    public:
    GfG()
    {
        i = 0;
        cout << "Inside Constructor\n";
    }

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

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

Output

```
Inside Constructor
End of main
Inside Destructor
```

You can clearly see the change in output. Now the destructor is invoked after the end of main. This happened because the scope of static object is through out the life time of program.

- **Static functions in a class**: Just like the static data members or static variables inside the class, static member functions also does not depend on object of class. We are allowed to invoke a static member function using the object and the '.' operator but it is recommended to invoke the static members using the class name and the scope resolution operator.**Static member functions are allowed to access only the static data members or other static member functions**, they can not access the non-static data members or member functions of the class.

```
// C++ program to demonstrate static
// member function in a class
#include<iostream>
using namespace std;

class GfG{
public:

    // static member function   static void printMsg()
    {
        cout<<"Welcome to GfG!";
    }
};

// main functionint main()
{
    // invoking a static member function    GfG::printMsg();
}
```

Output

```
Welcome to GfG!
```