STATIC KEYWORD IN C

Milan Murali

CONTENTS

- STATIC KEYWORD
  - STATIC VARIABLES
  - STATIC FUNCTIONS

Understanding the Static Keyword

- The **static** keyword in C is a storage class specifier that modifies the behavior of variables and functions.
- In C, storage classes determine the lifetime, scope, and initialization of a variable or function. There are several storage classes in C, and static is one of them.
- The **static** keyword in C is used to declare a variable or function that remains in memory throughout the program's execution.
- **Static** keyword in C restricts the scope of variables and functions to the file in which they are declared.

STORAGE CLASSES

- Storage classes in C define the characteristics of a variable or function, such as its **scope, visibility, lifetime, and memory location**. They essentially determine how a variable or function behaves within a program.
- Understanding storage classes is essential for effective memory management and ensuring the correct functioning of programs.

STORAGE CLASSES

- There are four types of storage classes in C
  - Automatic
  - External
  - Static
  - Register

Storage Class	Scope	Lifetime	Storage Location	Keyword	Default Value
Automatic	Block	Block execution	RAM	auto (optional)	Garbage Value
External	Global	Program execution	RAM	extern	Zero
Static	Block or file	Program execution	RAM	static	Zero
Register	Block	Block execution	Register	register	Garbage Value

Different Uses of Static Keyword

1. Static variables
2. Static functions

STATIC VARIABLES

Static Variables

- The **static** keyword is used to declare variables that have static storage class. They are initialized only once and retain their value between function calls.
- They have a global **scope** within the function they are defined in, but their visibility is limited to that function, and they persist for the entire **lifetime** of the program. Static variables are not accessible from other functions.
- **Static** variables are commonly used for various purposes such as maintaining state in functions or sharing data among multiple function calls.

Static Variables

Syntax -

```
static data_type variable_name;
```

Example -

```c
#include <stdio.h>

void count() {
    static int count = 0;
    count++;
    printf("Function called %d times.\n", count);
}

int main() {
    count();
    count();
    count();
    return 0;
}
```

Output -

```
Function called 1 times.
Function called 2 times.
Function called 3 times.
//retain value between function calls
```

STATIC FUNCTIONS

Static functions

- **Static** functions are functions that have a static storage class. They have internal linkage, which means they can only be called within the same source file. These functions are not available for external linkage, making them private to the file where they are defined.
- **Static** functions are useful for implementing encapsulation, hiding certain functions from other parts of the program.
- They enhance modularity by keeping the implementation details confined to a single file, making the code easier to manage and maintain.

Static functions

Syntax -

```
static return_type function_name(params) {

  // function body

}
```

Example -

```c
#include <stdio.h>

static void my_function() {
  printf("This is a static function.\n");
}

int main() {
  my_function();
  return 0;
}
```

KEY CHARACTERISTICS

- **Limited Scope :** Accessible only within the same source file.
- **Internal Linkage :** Cannot be referenced from other files.
- **Encapsulation :** To hide implementation details within a specific module.
- **Improving Code Organization:** By limiting function scope, you can enhance code readability and maintainability.
- **Avoiding Name Conflicts :** If you have functions with the same name in different files, making one static prevents conflicts.

Any questions?

THANKYOU!