`static` and `extern` in C

Pedram Pasandide

Introduction

In C, the storage class of a variable or function determines its scope, lifetime, and linkage. The `static` and `extern` keywords play a crucial role in controlling the visibility and accessibility of variables and functions across different files and functions.

`static` Keyword in C

The `static` keyword in C is used to define variables or functions with internal linkage or persistent storage duration. It has different meanings depending on where it is used.

Static Local Variables

A local variable declared with static inside a function retains its value between function calls. It is initialized only once (at program startup) and maintains its value across multiple function invocations.

```c
#include <stdio.h>

void counter() {
 static int count = 0;  // Retains its value between function
    calls
 count++;
 printf("Count: %d\n", count);
}

int main() {
 counter();
 counter();
 counter();
```

```
    return 0;
}
```

Output:

```
Count: 1
Count: 2
Count: 3
```

Without `static`, `count` would be reinitialized to `0` on every function call.

Static Global Variables

A global variable declared with `static` inside a file is only accessible within that file (internal linkage). This helps in encapsulating variables, avoiding naming conflicts across files.

`File1.c`:

```
// File1.c
#include <stdio.h>

static int globalVar = 10; // Not accessible outside this file

void printGlobal() {
    printf("Global Variable: %d\n", globalVar);
}
```

`File2.c`:

```
// File2.c
#include <stdio.h>
extern int globalVar;  // Error: globalVar is static in File1.c

int main() {
    printf("%d\n", globalVar); // This will cause a linker error
    return 0;
}
```

Static Functions

A function declared with `static` is only visible within the file it is defined in. This prevents the function from being accessed from other files.

```c
// File1.c
#include <stdio.h>

static void helper() {  // Only accessible within this file
 printf("This is a static function\n");
}

void callHelper() {
 helper();
}
```

If another file tries to call `helper()`, it will result in a linker error.

Static Arrays in Function Parameters (C99 and Later)

When using `static` in function parameters, it ensures that the passed array has at least the specified size.

```c
void processArray(int arr[static 10]) {
 // Assumes arr has at least 10 elements
 printf("First element: %d\n", arr[0]);
}
```

The `static` keyword in the array parameter ensures the function expects an array of at least `10` elements. If an array of fewer than `10` elements is passed, it could lead to errors.

`extern` Keyword in C

The `extern` keyword is used to declare a global variable or function that is defined in another file or outside the current scope.

Extern Global Variables

Allows sharing global variables across multiple files. The `extern` keyword only declares the variable, it does not allocate memory for it.

`File1.c`:

```
// File1.c
#include <stdio.h>

int sharedVar = 42;  // Global variable definition
```

`File2.c`:

```
// File2.c
#include <stdio.h>

extern int sharedVar;  // Referencing the variable from File1.c

int main() {
 printf("Shared Variable: %d\n", sharedVar);
 return 0;
}
```

Extern Functions

Functions are `extern` by default, meaning they can be called from other files without explicitly using `extern`.

`File1.c`:

```
// File1.c
#include <stdio.h>

void sayHello() {
 printf("Hello from File1!\n");
}
```

`File2.c`:

```
// File2.c
#include <stdio.h>
```

```c
extern void sayHello();  // Function declaration

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

Extern vs Static

Here is summary of what we discuss:

- `static` is used for data hiding, persistent local variables, and file-local functions.
- `extern` is used to access global variables and functions across multiple files.
- `static` functions and variables are restricted to the file they are defined in.
- `extern` variables and functions must be defined elsewhere but can be accessed globally.

Table 1: Summary of Extern vs Static

Feature	`static`	`extern`
Storage Duration	Global or Local	Global
Scope	Limited to the file	Accessible from other files
Initialization	Only once	Only once (in the defining file)
Use Case	Data encapsulation	Sharing across files