Understanding Storage Classes in C

Storage classes define **scope**, **lifetime**, and **memory location** of variables/functions. They help optimise memory usage and control access.

1. Storage Classes in C

Storage Class	Keyword	Lifetime	Scope	Default Value	Memory Segment
Automatic	`auto`	Function block	Local	Garbage	Stack
Register	`register`	Function block	Local	Garbage	CPU Register (if possible)
Static	`static`	Entire program	Local/ Global	Zero	`.data` or `.bss`
External	`extern`	Entire program	Global	Zero	`.data` or `.bss`
Mutable	(N/A in C)	-	-	-	-

2. Detailed Explanation with Examples

(1) `auto` (Automatic)

- **Default for local variables**.
- **Lifetime**: Until the function exits.

- **Memory**: Stack (destroyed after function ends).

```
void func() {
    auto int x = 10;  // Same as `int x = 10;`
    printf("%d", x);  // Valid
}
// `x` is destroyed here
```

(2) `register` (Hint for CPU Registers)

- **Suggests** the compiler to store the variable in a **CPU register** (faster access).
- **No address can be taken** (`&x` is invalid).
- **Compiler may ignore** if registers are unavailable.

```
void func() {
    register int i;
    for (i = 0; i < 1000; i++) {  // Faster access (if in register)
        printf("%d", i);
    }
}
```

(3) `static` (Persistent Storage)

A. Static Local Variables

- **Lifetime**: Entire program.
- **Memory**: `.data` (if initialized) or `.bss` (if uninitialized).
- **Scope**: Only inside the function.

```
void counter() {
    static int count = 0;  // Retains value between calls
    count++;
    printf("%d\n", count);
}
int main() {
    counter();  // 1
    counter();  // 2
    return 0;
}
```

B. Static Global Variables/Functions

- **Scope**: Limited to the file where declared.

- **Prevents linkage to other files**.

```c
// File1.c
static int hidden = 42;  // Only visible in File1.c
static void secret() {   // Only visible in File1.c
    printf("Secret!\n");
}
```

(4) `extern` (Global Variable Sharing)

- **Used to share variables/functions across files**.
- **Lifetime**: Entire program.
- **Memory**: `.data` or `.bss`.

```c
// File1.c
int global = 100;  // Defined in File1.c

// File2.c
extern int global;  // Declared (not defined)
void print_global() {
    printf("%d\n", global);  // Accesses `global` from File1.c
}
```

3. Key Takeaways

Storage Class	When to Use?	Key Behavior
`auto`	Local variables (default)	Dies after function ends
`register`	Frequently used variables (loops)	Faster access (if optimized)
`static`	Preserve state between calls / hide globals	Lives forever, scope-limited

`extern`	Share globals across files	Defined once, declared elsewhere

4. Memory Mapping of Storage Classes

Storage Class	Memory Segment	Example
`auto`	Stack	`int x;`
`register`	CPU Register (or stack)	`register int i;`
`static`	`.data` (initialized) / `.bss` (uninitialized)	`static int c;`
`extern`	`.data` or `.bss` (shared)	`extern int global;`

5. Common Interview Questions

1. **What happens if you declare a `static` variable inside a function?**
   → It retains its value between function calls.
2. **Can `extern` variables be initialized during declaration?**
   → No, `extern` only declares (does not allocate memory).
3. **Why can't we take the address of a `register` variable?**
   → Registers don't have memory addresses.
4. **What's the difference between `static` and `global` variables?**
   → `static` limits scope to the file, while `global` is accessible across files.

6. Practical Tips

✅ **Use** `static` for:

- Maintaining state between function calls.
- Hiding variables/functions from other files.

✅ **Use** `extern` for:

- Sharing variables across multiple files.

🚫 **Avoid** `register` in modern code (compilers optimize better).