Storage Classes

Introduction

- The scope of a variable is the region of the program within which the variable can be used (accessible, active, visible).
- The longevity of a variable refers to the period (lifetime) during which a variable retains an assigned value during execution of the program (alive, existence, lifetime).
- Storage class specifier indicates
  - **Place of Storage**: where the variables would be stored
  - **Scope**: what would be their region of usability (visibility, accessibility)?
  - **Lifetime**: how long they would exist (Existence)?
  - **Default values**: what would be the default values at the time of declaration of variables?
- Standard C provides four storage class specifiers, they are automatic, register, static, extern.
- The storage class specifier precedes the declaration of a variable.
  - ➢ **The general syntax: storage_class_specifier data_type variable_name ;**
- Example: register float price; static double area;

1. auto Storage Class

- Automatic variables are declared in a function header, in a for loop, or in a { } block.
- Automatic variables are created (memory allocated) when the function is called and destroyed (freed memory) automatically when the function is exited.
- Automatic variables are private or local to the function, loop or block in which they are declared, Hence, the Automatic variables are also called local or internal variables.

✍ A variable declared inside a function without storage class specifier is an automatic variable, by default.

Example: ***auto double area;*** or simply ***double area;***

- **Keyword** used: auto
- **Place of storage**: Main Memory (RAM)
- **Scope**: Within the function, the loop or the block where it is declared.
- **Lifetime**: Exists from the time of entry in the function, the loop, or the block to its return to the calling function, to the end of loop, or to the end of block.
- **Default value**: Garbage value (previously someone used value)

2. register Storage Class

- Values stored in CPU Registers are accessed faster than those stored in the main memory (Primary memory or RAM).
- The register storage class specifier is used to allow faster access of variables.

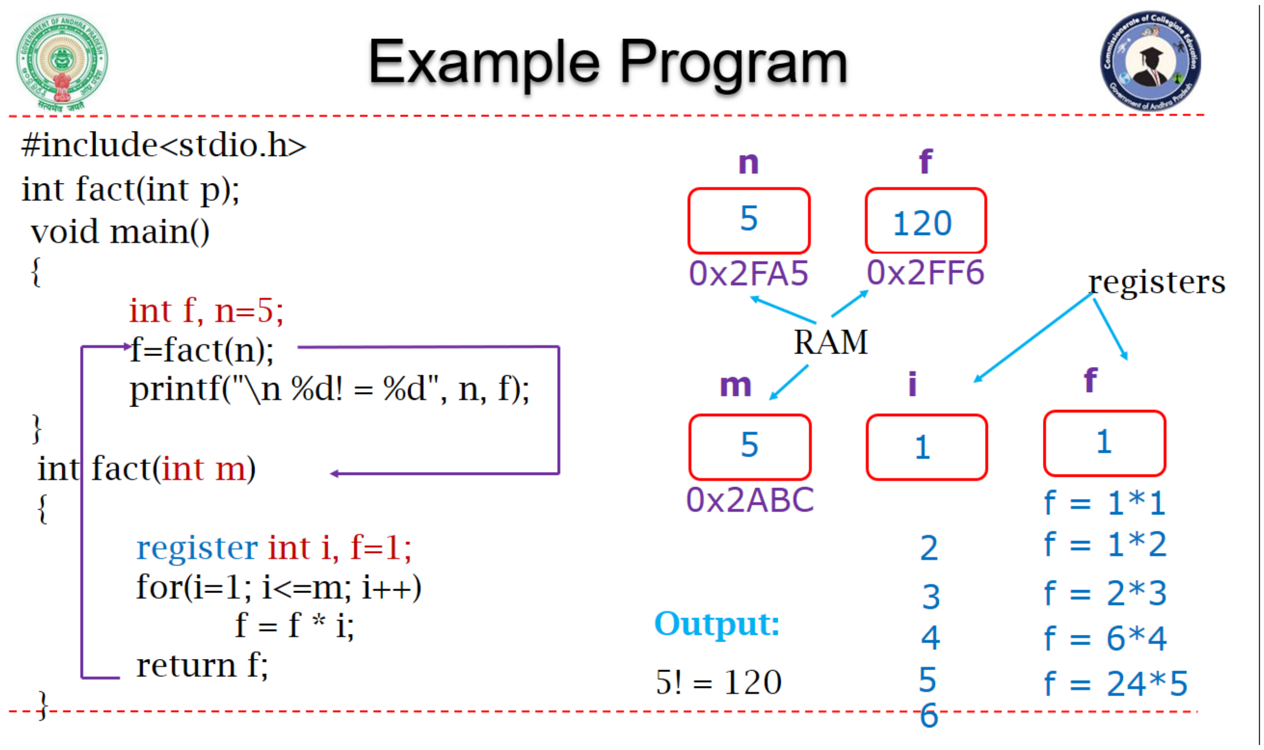
✍ Using register storage class specifier, programmer can only suggest the compiler to store those variables in the CPU registers (if registers are available, otherwise stores in main memory) which are used frequently or whose access times are critical.

✍ We cannot access address of a register variable using "&" operator.

✍ We cannot apply register storage class on global (external) variables.

🗐 Example: ***register double area;***

- **Keyword** used: register
- **Place of storage**: CPU Registers
- **Scope**: Within the block or the function where it is declared.
- **Lifetime**: Exists from the time of entry in the function, the loop, or the block to its return to the calling function, to the end of loop, or to the end of block.
- **Default value**: Garbage value (previously someone used value)



Example Program

```
#include<stdio.h>
int fact(int p);
 void main()
 {
        int f, n=5;
        f=fact(n);
        printf("\n %d! = %d", n, f);
 }
 int fact(int m)
 {
         register int i, f=1;
         for(i=1; i<=m; i++)
                 f = f * i;
         return f;
 }
```

Output:

```
5! = 120
```

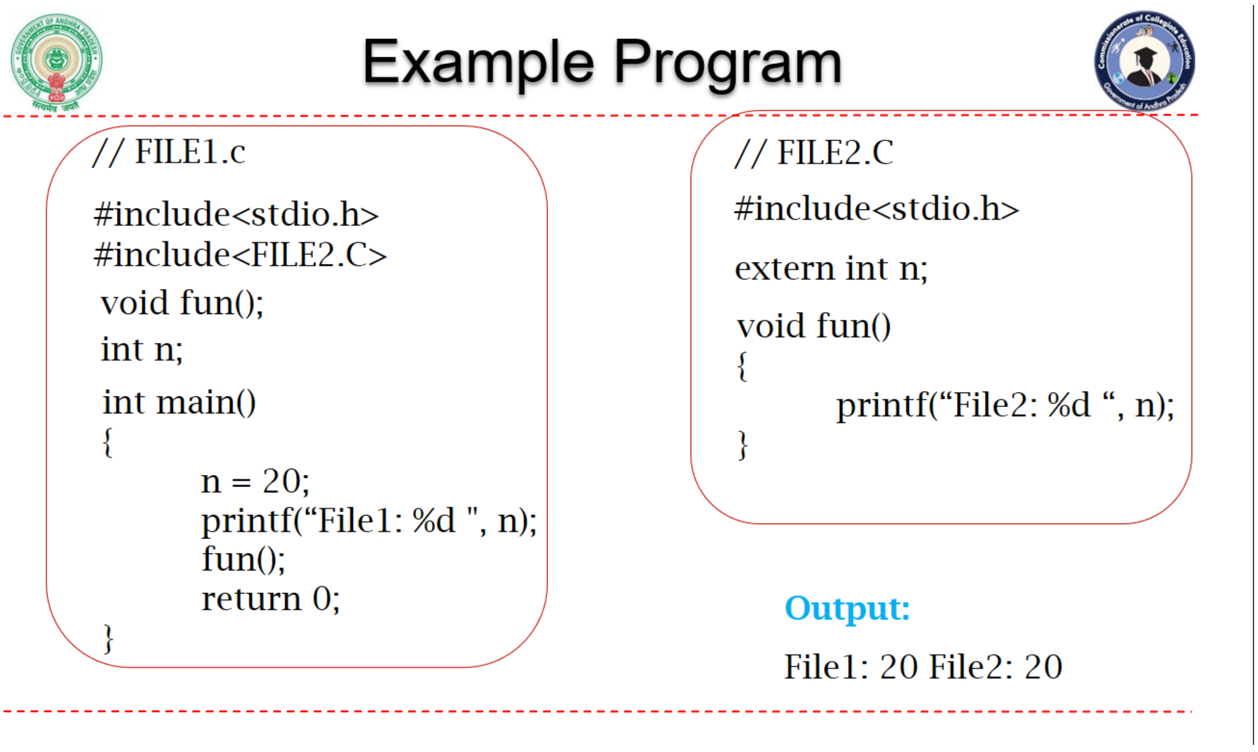
3. extern Storage Class

- In software development, a large program can be broken down into smaller programs, each program keeps in separate file and compiled separately for ease of understanding and modification.
- However, these smaller programs may need to share certain variables for processing. In such situations C language provides an extern storage class.

✍ When there are multiple files in a program and we need to use a particular variable or function in a file apart from which it is declared, then use the extern storage class.

🗐 Example: ***extern double area;***

- **Keyword** used: extern
- **Place of storage**: Main Memory (RAM)
- **Scope**: Within all program files that are a part of the program.
- **Lifetime**: Exists throughout the execution of a program.
- **Default value**: 0 (zero)



Example Program

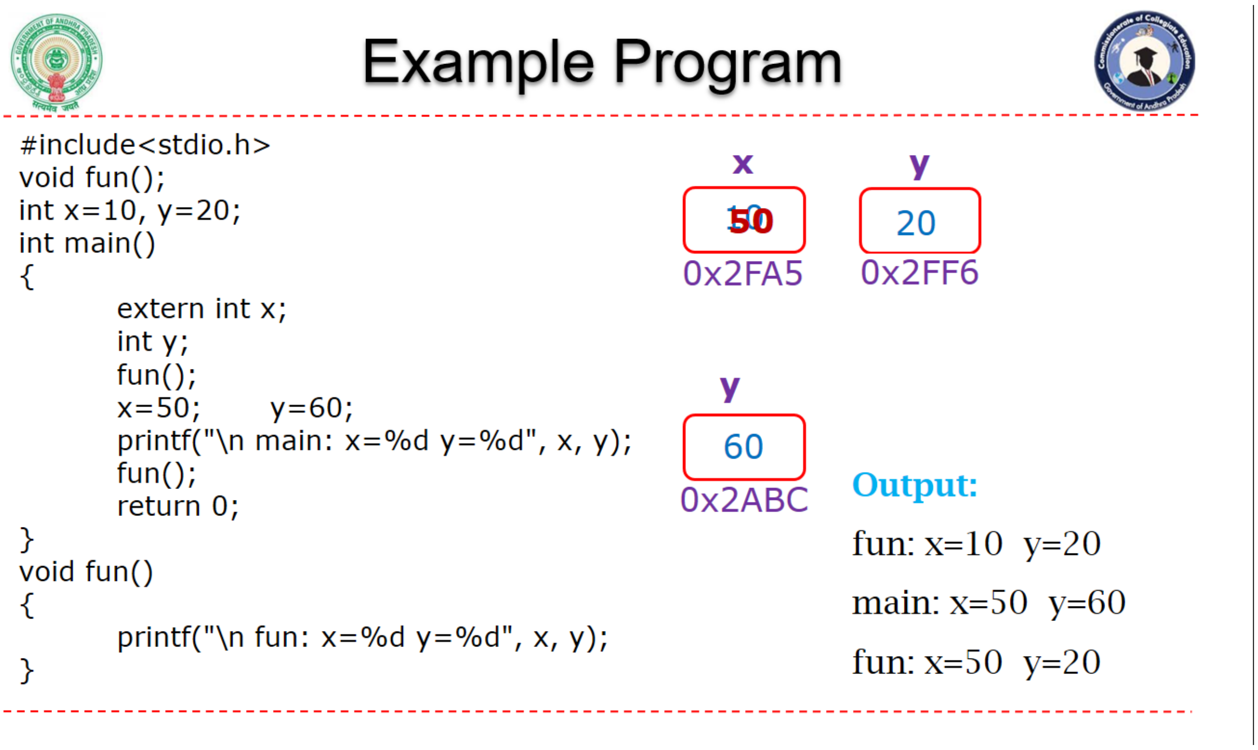
```
// FILE1.c

#include<stdio.h>
#include<FILE2.C>
void fun();
int n;
int main()
{
        n = 20;
        printf("File1: %d ", n);
        fun();
        return 0;
}
```

```
// FILE2.C
#include<stdio.h>
extern int n;
void fun()
{
        printf("File2: %d ", n);
}
```

Output:

```
File1: 20 File2: 20
```



Example Program

```
#include<stdio.h>
void fun();
int x=10, y=20;
int main()
{
        extern int x;
        int y;
        fun();
        x=50;       y=60;
        printf("\n main: x=%d y=%d", x, y);
        fun();
        return 0;
}
void fun()
{
        printf("\n fun: x=%d y=%d", x, y);
}
```

x
~~10~~ 50
0x2FA5
y
20
0x2FF6
y
60
0x2ABC

Output:

```
fun: x=10  y=20
main: x=50  y=60
fun: x=50  y=20
```

4. static Storage Class

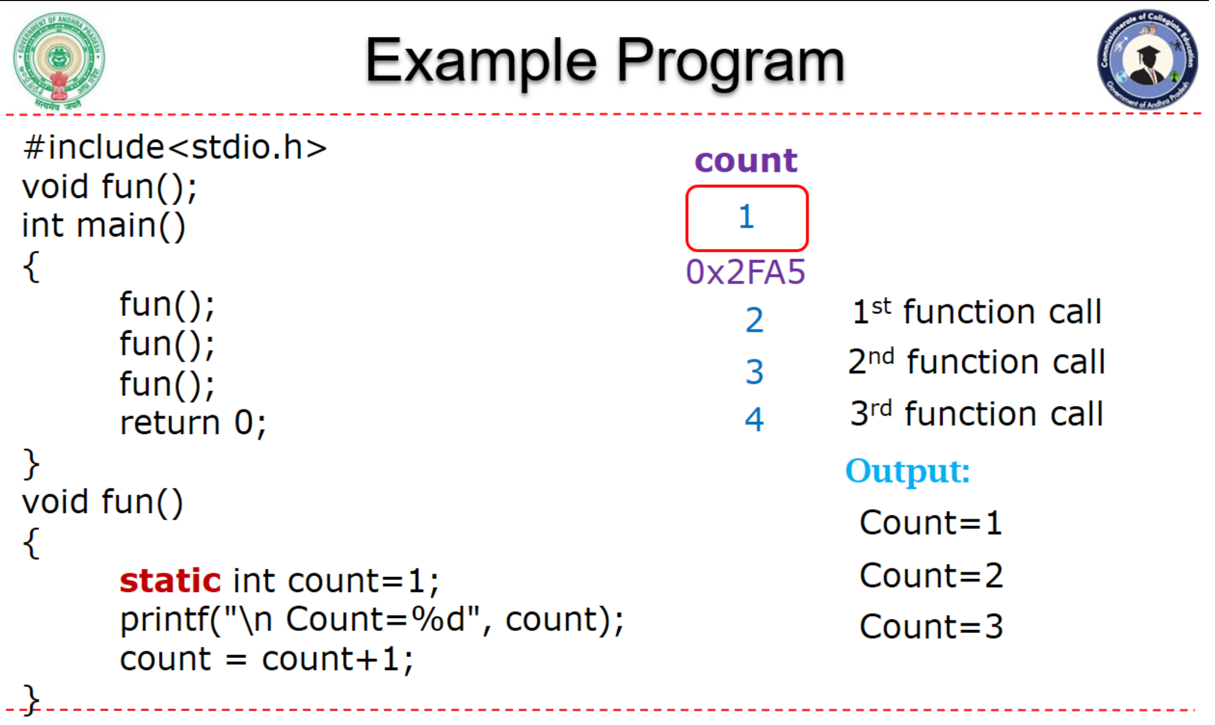
- The static storage class, applied to a local variable, a global variable, or a function.
- The variables declared with **static** storage class are automatically initialized to zero if programmer is not assigned at the time of variable declaration.
- Local (internal) static variables are local to a particular function just as automatic variables are, but unlike automatics, they remain in existence and retains value between function calls.
- Making global variable and functions as file scope from program scope.

✍ Global variable (declared outside of all functions) are visible in any part (file) of the entire program. If a global variable is declared with **static** storage class, then the variable is invisible outside of the file in which it is declared.

✍ Normally, functions are global (by default), visible to any part (file) of the entire program. If a function is declared with **static** storage class, then the function is invisible outside of the file in which it is declared.

🗐 Example: ***static double area;***

- **Keyword** used: static
- **Place of storage**: Main Memory (RAM)
- **Scope**:
  - ✓ **Local**: Within the block or function where it is declared.
  - ✓ **Global**: Within the program in which it is declared.
- **Lifetime**: Exists throughout the execution of a program but invisible outside of a file.
  - ✓ Local: Retain value between function calls or block entries.
  - ✓ Global: Preserve value in program files.
- **Default value**: 0 (zero)



Example Program

```
#include<stdio.h>
void fun();
int main()
{
        fun();
        fun();
        fun();
        return 0;
}
void fun()
{
        static int count=1;
        printf("\n Count=%d", count);
        count = count+1;
}
```

count

$\boxed{1}$

0x2FA5

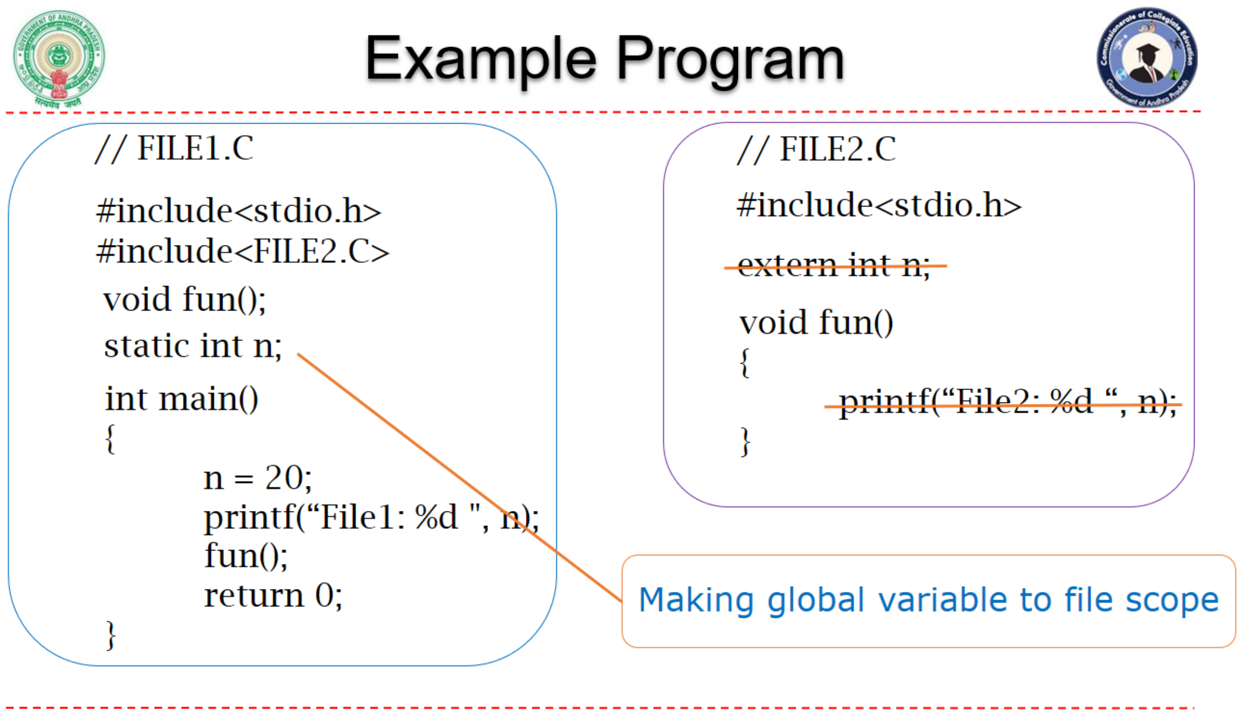
2   1st function call

3   2nd function call

4   3rd function call

Output:

```
Count=1
Count=2
Count=3
```



Example Program

```
// FILE1.C

#include<stdio.h>
#include<FILE2.C>
void fun();
static int n;
int main()
{
        n = 20;
        printf("File1: %d ", n);
        fun();
        return 0;
}
```

```
// FILE2.C
#include<stdio.h>
```

~~extern int n;~~

```
void fun()
{
```

~~printf("File2: %d ", n);~~

```
}
```

Making global variable to file scope

Summary

Storage Class Specifier	Features			
	Place of Storage	**Scope / Accessibility**	**Lifetime / Existence**	**Default Value**
auto	Main Memory	Within the block or function where it is declared.	Exists from the time of entry in the function or block to its return to the calling function or to the end of block.	Garbage
register	CPU Register	Within the block or function where it is declared.	Exists from the time of entry in the function or block to its return to the calling function or to the end of block.	Garbage
static	Main Memory	**Local**: Within the block or function where it is declared. **Global**: Within the program in which it is declared	**Local**: Retain value between function calls or block entries. **Global**: Preserve value in program files.	Zero
extern	Main Memory	Within all program files that are a part of the program.	Exists throughout the execution of a program	Zero

References Books

1. E Balagurusamy – Programming in ANSIC – Tata McGraw-Hill Publications
2. Reema Thareja – Introduction to C Programming – Oxford University Press
3. Pradip Dey, Manas Ghosh – Programming in C – Oxford University Press
4. Brain W Kernighan, Dennis M Ritchie – The 'C' Programming Language – Pearson
5. Jeri R Hanly, Elliot B Koffman – Problem Solving and Program Design in C – Pearson

References Links

1. https://nptel.ac.in/courses/106/105/106105171/
2. https://nptel.ac.in/courses/106/104/106104128/
3. http://www.cplusplus.com/doc/tutorial/
4. https://en.cppreference.com/w/c