Storage Class in C

A storage class defines the scope (visibility) and lifetime of variables and/or functions within a C Program. They precede the type that they modify. We have four different storage classes in a C program −

- auto
- register
- static
- extern

Storage Class	Declaration Location	Scope (Visibility)	Lifetime (Alive)
auto	Inside a function/block	Within the function/block	Until the function/block completes
register	Inside a function/block	Within the function/block	Until the function/block completes
extern	Outside all functions	Entire file plus other files where the variable is declared as extern	Until the program terminates
static (local)	Inside a function/block	Within the function/block	Until the program terminates
static (global)	Outside all functions	Entire file in which it is declared	Until the program terminates

- **auto:**
  - The auto storage class is the default storage class for all local variables.
  - Auto can only be used within functions, i.e., local variables.
  - They are assigned a garbage value by default whenever they are declared.

```
{
  int  mount;
  auto int month;
}
```

- **register:**
  - This storage class declares register variables which have the same functionality as that of the auto variables. The only difference is that the compiler tries to store these variables in the register of the microprocessor if a free register is available. If a free register is not available, these are then stored in the memory only.
  - Few variables which are to be accessed very frequently in a program are declared with the register keyword to execute much faster, which improves the running time of the program. The register should be used for variables like *counters* which require quick access.
  - The register variable has a maximum size equal to the register size of microprocessor and cannot have the unary '&' operator applied to it because it does not have a memory location. So, we cannot obtain the address of a register variable using pointers.
  - NOTE: It should also be noted that defining 'register' does not mean that the variable will be stored in a register. It means that it might be stored in a register depending on hardware and implementation restrictions.

```
{
  register int miles;
}
```

- **static:**
  - **Local Static Variable:**
    - The static storage class instructs the compiler to preserve their value even after they are out of their scope.
    - Hence, static variables preserve the value of their last use in their scope. That is why are initialized only once and exist till the termination of the program. Thus, no new memory is allocated because they are not re-declared.
    - Their scope is local to the function to which they were defined.
  - **Global Static Variable:**
    - A global static variable is one that can only be accessed in the file where it is created. This variable is said to have file scope.
    - Means global static variable's scope to be restricted to the file in which it is declared.

- **Static Function:**
  - A static function is a function that has a scope that is limited to its object file.
  - This means that the static function is only visible in its object file and cannot be used by other object files.

```c
#include <stdio.h>


/* function declaration */
static void staticFun(void);
void func(void);


static int count = 5; /* global variable */

main()
{

   while(count--)
   {
      func();
   }

   staticFun();

   return 0;
}


/* function definition */
void func( void ) {

   static int i = 5; /* local static variable */
   i++;

   printf("i is %d and count is %d\n", i, count);
}


/* static function definition */
static void staticFun( void )
{
   printf("This is Static Function\n”);
}
```

```
EXPECTED OUTPUT:

i is 6 and count is 4
i is 7 and count is 3
i is 8 and count is 2
i is 9 and count is 1
i is 10 and count is 0

This is Static Function
```

- **extern:**
  - When we have multiple files and we define a global variable or function, which will also be used in other files, then *extern* will be used in another file to provide the reference of defined variable or function i.e. *extern* is used to declare a global variable or function in another file.
  - It is most commonly used when there are two or more files sharing the same global variables or functions.
  - Basically, the value is assigned to it in a different block and this can be overwritten/changed in a different block as well. So, an extern variable is nothing but a global variable initialized with a legal value where it is declared in order to be used elsewhere.
  - It can be accessed within any function/block.

First File: main.c

```c
#include <stdio.h>

int count ;

extern void write_extern();

main()
{
  count = 5;

  write_extern();
}
```

Second File: support.c

```
#include <stdio.h>

extern int count;

void write_extern(void)
{
   printf("count is %d\n", count);
}
```

Output

```
Count is 5
```