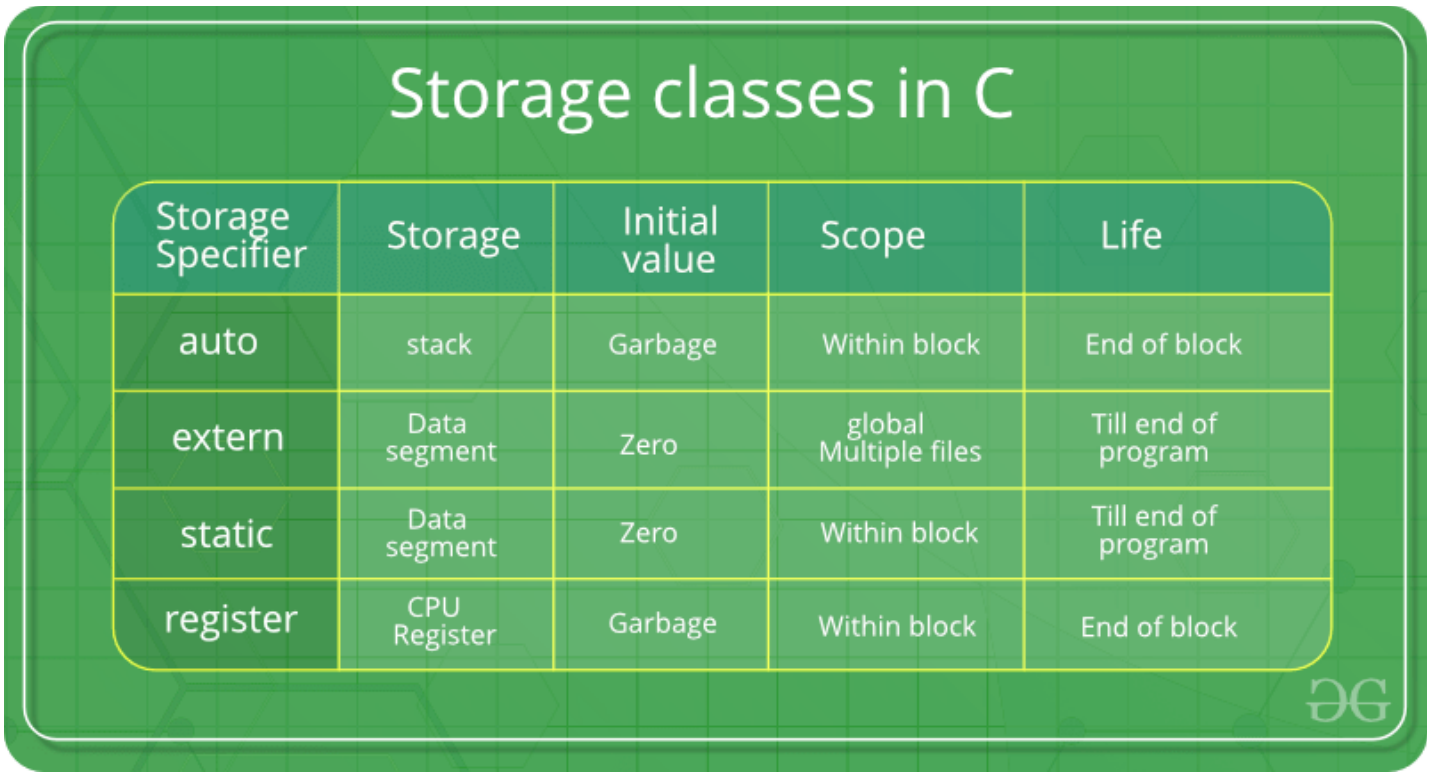

Storage Classes in C

Storage Classes are used to describe the features of a variable/function. These features basically include the scope, visibility and life-time which help us to trace the existence of a particular variable during the runtime of a program.

C language uses 4 storage classes, namely:

auto: This is the default storage class for all the variables declared inside a function or a block. Hence, the keyword auto is rarely used while writing programs in C language. Auto variables can be only accessed within the block/function they have been declared and not outside them (which defines their scope). Of course, these can be accessed within nested blocks within the parent block/function in which the auto variable was declared. However, they can be accessed outside their scope as well using the concept of pointers given here by pointing to the very exact memory location where the variables resides. They are assigned a garbage value by default whenever they are declared.

Extern

Extern storage class simply tells us that the variable is defined elsewhere and not within the same block where it is used. 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. Also, a normal global variable can be made extern as well by placing the 'extern' keyword before its declaration/definition in any function/block. The main purpose of using extern variables is that they can be accessed between two different files which are part of a large program

1. A declaration can be done any number of times but definition only once.
2. "extern" keyword is used to extend the visibility of variables/functions().
3. Since functions are visible throughout the program by default. The use of extern is not needed in function declaration/definition. Its use is redundant.
4. When extern is used with a variable, it's only declared not defined.
5. As an exception, when an extern variable is declared with initialization, it is taken as the definition of the variable as well.

Static

This storage class is used to declare static variables which are popularly used while writing programs in C language. Static variables have a property of preserving their value even after they are out of their scope! Hence, static variables preserve the value of their last use in their scope. So we can say that they 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 variables can be accessed anywhere in the program. By default, they are assigned the value 0 by the compiler.

In C, static variables can only be initialized using constant literals

static int i = initializer(); is not possible

If we will use the static keyword with a global variable then the scope of a global variable limited to the file in which it declares. in other word we can say that static makes a variable or function private for the file in which it declares.

When we have initialized a static variable then it will be created in the Data Segment (.ds) either it will be created in .bss (block started symbol) of the process.

If we have used the static keyword with a variable or function, then only internal or none linkage is worked

Note: *static variable preserves its previous value and it is initialized at compilation time when memory is allocated. If we do not initialize the static variable, then it's a responsibility of compiler to initialize it with Zero value.*

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. This makes the use of register variables to be much faster than that of the variables stored in the memory during the runtime of the program. If a free register is not available, these are then stored in the memory only. Usually few variables which are to be accessed very frequently in a program are declared with the register keyword which improves the running time of the program. An important and interesting point to be noted here is that we cannot obtain the address of a register variable using pointers.

Registers are faster than memory to access, so the variables which are most frequently used in a C program can be put in registers using *register* keyword. The keyword *register* hints to compiler that a given variable can be put in a register. It's compiler's choice to put it in a register or not. Generally, compilers themselves do optimizations and put the variables in register.

If you use & operator with a register variable then compiler may give an error or warning (depending upon the compiler you are using), because when we say a variable is a register, it may be stored in a register instead of memory and accessing address of a register is invalid.

```
int main()
{
register int i = 10;
int *a = &i;
printf("%d", *a);
getchar();
return 0;
}
```

register keyword can be used with pointer variables. Obviously, a register can have address of a memory location. There would not be any problem with the below program.

```
int main()
{
register int i = 10;
register int *a = &i;
printf("%d", *a);
getchar();
return 0;
}
```

Register variable stored in CPU register instead of memory and its properties is generally similar to the local variable. Accessing of the register is faster than the memory. So generally, the register variable is used in looping.
A register keyword only gives the indication to the compiler to store this variable in the register instead of RAM, But it totally depends on the compiler. The compiler decides where to put the variable in register or RAM.

Note: We can not use & and * operator with a register variable because access the address of the register variable is invalid.

Global

Variables which declared outside the function are called global variables. A global variable is not limited to any function or file it can be accessed by any function or outside of the file. If you have not initialized the global variables, then it automatically initialized to 0 at the time of declaration.

1. Unlike local variables, global variables are not destroyed as soon as the function ends.

2. Global variables are initialized as 0 if not initialized explicitly.

3. Initialized global variable creates in DS and uninitialized global variable creates in BSS.

4. By default, all global variable has external linkage.