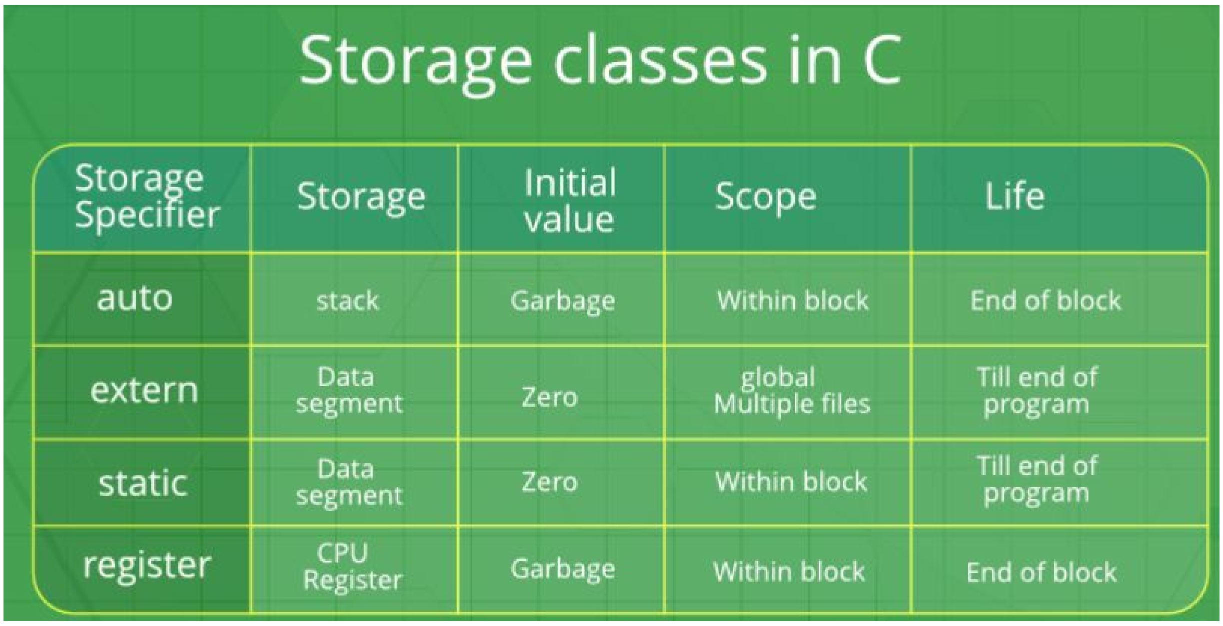
Storage class

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. It deals with following factors.

- The variable scope.
- The location where the variable will be stored.
- The initialized value of a variable.
- A lifetime of a variable.
- Who can access a variable?

auto	It is a default storage class.
extern	It is a global variable.
static	It is a local variable which is capable of returning a value even when control is transferred to the function call.
register	It is a variable which is stored inside a Register.

Summary



Storage classes in C

Storage Specifier	Storage	Initial value	Scope	Life
auto	stack	Garbage	Within block	End of block
extern	Data segment	Zero	global Multiple files	Till end of program
static	Data segment	Zero	Within block	Till end of program
register	CPU Register	Garbage	Within block	End of block

auto: The variables defined using auto storage class are called as local variables. Auto stands for automatic storage class. A variable is in auto storage class by default if it is not explicitly specified.

The scope of an auto variable is limited with the particular block only. Once the control goes out of the block, the access is destroyed. This means only the block in which the auto variable is declared can access it.

A keyword auto is used to define an auto storage class. By default, an auto variable contains a garbage value.

Example

```
#include <stdio.h>
int main( )
{
  auto int j = 1;
  {
    auto int j= 2;
    {
      auto int j = 3;
      printf ( " %d ", j);
    }
    printf ( "\t %d ",j);
  }
  printf( "%d\n", j);}

output:  3 2 1
```

extern: Extern stands for external storage class. Extern storage class is used when we have global functions or variables which are shared between two or more files.

Keyword **extern** is used to declaring a global variable or function in another file to provide the reference of variable or function which have been already defined in the original file.

The variables defined using an extern keyword are called as global variables. These variables are accessible throughout the program. Notice that the extern variable cannot be initialized it has already been defined in the original file

```
#include <stdio.h>
extern i;
main() {
   printf("value of the external integer is = %d\n", i);
   return 0;}
```

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.

Static local variable is a local variable that retains and stores its value between function calls or block and remains visible only to the function or block in which it is defined.

Static global variables are global variables visible only to the file in which it is declared.

```
#include <stdio.h> /* function declaration */
void next(void);
static int counter = 7; /* global variable */
main() {
 while(counter<10) {
      next();
      counter++;   }
return 0;}
void next( void ) {    /* function definition */
   static int iteration = 13; /* local static variable */
   iteration ++;
   printf("iteration=%d and counter= %d\n", iteration, counter);}
```

output

```
iteration=14 and counter= 7
iteration=15 and counter= 8
iteration=16 and counter= 9
```

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.

The keyword **register** is used to declare a register storage class. The variables declared using register storage class has lifespan throughout the program.

It is similar to the auto storage class. The variable is limited to the particular block. The only difference is that the variables declared using register storage class are stored inside CPU registers instead of a memory. Register has faster access than that of the main memory.

The variables declared using register storage class has no default value. These variables are often declared at the beginning of a program.

```
#include <stdio.h> /* function declaration */
main() {
{register int  weight;
int *ptr=&weight ;/*it produces an error when the compilation occurs ,we cannot get a
memory location when dealing with CPU register*/}
}
```

NOTE:

- A storage class is used to represent additional information about a variable.
- Storage class represents the scope and lifespan of a variable.
- It also tells who can access a variable and from where?
- Auto, extern, register, static are the four storage classes in 'C'.
- auto is used for a local variable defined within a block or function
- register is used to store the variable in CPU registers rather memory location for quick access.
- Static is used for both global and local variables. Each one has its use case within a C program.
- Extern is used for data sharing between C project files.

Scope of a variable

Local scope

A local scope or block is collective program statements put in and declared within a function or block (a specific region enclosed with curly braces) and variables lying inside such blocks are termed as local variables. All these locally scoped statements are written and enclosed within left ({) and right braces (}) curly braces. There's a provision for nested blocks also in C which means there can be a block or a function within another block or function. So it can be said that variable(s) that are declared within a block can be accessed within that specific block and all other inner blocks of that block, but those variables cannot be accessed outside the block.

```
#include <stdio.h>

int main ()
{
    /* local variable definition and initialization */    int x,y,z;

    /* actual initialization */    x = 20;
    y = 30;
    z = x + y;

    printf ("value of x = %d, y = %d and z = %d\n", x, y, z);

    return 0;
}
```

Global Variable

Global variables are defined outside a function or any specific block, in most of the case, on the top of the C program. These variables hold their values all through the end of the program and are accessible within any of the functions defined in your program.

Any function can access variables defined within the global scope, i.e., its availability stays for the entire program after being declared.

```
#include <stdio.h>

/* global variable definition */int z;

int main ()
{
    /* local variable definition and initialization */    int x,y;

    /* actual initialization */    x = 20;
    y = 30;
    z = x + y;

    printf ("value of x = %d, y = %d and z = %d\n", x, y, z);

    return 0;

}
```