Storage classes in C

In this tutorial you will learn everything about storage classes in C and different classifications of storage classes like auto, register, static and extern explained with the use of some simple examples.

What are storage classes in c?

Storage classes in C describe the properties of the address where a variable or function is stored. This means storage classes are used to determine the location, scope or lifetime, and initialized value of a variable. RAM memory, CPU registers are the location where these values are generally saved for the future scope and visibility of the program. In C programming language storage class can determine the lifetime of a variable by classifying it as 'global' or 'local'. They are basically of four storage classes:

1. auto
2. register
3. static
4. xtern

There are also storage class specifiers related to every class. The job of any storage class is to tell or order the compiler- '**How to store the specific variable or function**'. The declaration syntax of the specifiers are as follows:

```
storage_class_specifier variable_type variable name;
```

You can declare only one of the storage class specifiers in a single declaration statement.
In the absence of storage class specifiers memory allocation happens by the following rules:

1. Variables declared within the function are considered as 'auto'.
2. Functions declared inside the function are considered to be 'extern'.
3. Variables and functions that are declared outside a function are categorized as 'static' having 'external linkage'.

File scope variables that are 'static' with external linkage can be used only inside the subsequent file. Whereas, the ones with 'external linkage' are freely available to all the files within the program. Local variables are one of their kind because they don't have any type of linkage. So they can be used only inside the block of coding where they have been declared.

Auto storage class in c

A variable declared with the 'auto' specifier has a place with the automatic storage class. All types or variables without any specifier are considered to be of this kind by default. The scope and visibility of an automatic variable are constrained to the function in which it is defined. So on exiting the function block the memory allocated for the automatic variable gets freed. From this, we can infer that the memory allocation of automatic variables is done dynamically. When considering the initialization part, by default, the automatic variable in c is initialized assigned to a garbage value.

Consider the following program:

```
#include <stdio.h>

main() {
  auto int a = 1; {
    auto int a = 2; {
      auto int a = 3;
      printf("\n%d ", a);
    }
    printf("%d ", a);
  }
  printf("%d", a);
}
```

The output is '3 2 1'.

In this program an integer variable is defined three times with the same name '**a**', still, the compiler shows no error. This is because the existence of '**a=3**' remains only within the third block. As soon as the execution is complete and the second layer of the block(where a=2) takes the control, the name of the variable 'a' or 'a=3' ceases to exist. Now 'a' is an all-new identifier and you are free to specify any value you want. Thus, the cycle goes on and it prints '3 2 1' using the same variable 'a'.

Register storage class in c

The Register storage class belongs to the genre of automatic ones as here the declared storage class will function only with the block too where it has been declared. The difference lies in the speed of access. We know that computer memory or RAM works a bit slower than the registers, which have a small storage capacity. So declaring the storage class as 'register' tells the compiler to store it inside the CPU registers, and they are going to be accessed frequently in the future. But the drawback is that only a few selected variables can be stored in this storage class and you can not access the exact address location (using '&'). Another **limitation** is that no initial value can be assigned but the use of this storage class decreases the compilation time significantly.

Static storage class in c

The static storage class in C is used to declare static variables that get allocated to an initial value **zero**. The keyword specifier for the static variable is **static**. Though these variables can't be used outside the function it has been defined but the most important feature is it does not lose its value even if the execution of the block completed, rather hold it till the main function is in operation.

```
#include <stdio.h>

int show() {
  int a = 10;
  static int s = 10;
  printf("a=%d\t s=%d\n", a, s);
  a = a + 1;
  s = s + 1;
}

main() {
  show();
  show();
  show();
}
```

Output:

```
a=10   s=10
a=10   s=11
a=10   s=12
```

So every time during the execution of the 'show()' value of static variables prints the value as per the last assigned value of each cycle, not the initial one. Whereas the auto storage class defines every time 'a' as 10 and it gets printed accordingly.

External storage class in c

Any Variables belonging to an external storage class can be defined in a block and used in another block or segment simultaneously. The extern specifier gives an indication to the compiler that there exists an external linkage.Thus, it works like a global variable with an initially assigned value. It has the capability of working in two different files inside a large program. It is defined by the keyword 'extern' like
extern int x;
extern show();