Storage classes in C

In C language, each variable has a storage class which decides the following things:

- **scope** i.e where the value of the variable would be available inside a program.
- **default initial value** i.e if we do not explicitly initialize that variable, what will be its default initial value.
- **lifetime** of that variable i.e for how long will that variable exist.

The following storage classes are most oftenly used in C programming,

1. **Automatic variables**
2. **External variables**
3. **Static variables**
4. **Register variables**

Automatic variables: `auto`

Scope: Variable defined with `auto` storage class are local to the function block inside which they are defined.

Default Initial Value: Any random value i.e garbage value.

Lifetime: Till the end of the function/method block where the variable is defined.

A variable declared inside a function without any storage class specification, is by default an **automatic variable**. They are created when a function is called and are destroyed **automatically**when the function's execution is completed. Automatic variables can also be called **local variables**because they are local to a function. By default they are assigned **garbage value** by the compiler.

```
#include<stdio.h>


void main()
{
    int detail;
    // or
    auto int details;    //Both are same
}
```

External or Global variable

Scope: Global i.e everywhere in the program. These variables are not bound by any function, they are available everywhere.

Default initial value: 0(zero).

Lifetime: Till the program doesn't finish its execution, you can access global variables.

A variable that is declared outside any function is a **Global Variable**. Global variables remain available throughout the program execution. By default, initial value of the Global variable is 0(zero). One important thing to remember about global variable is that their values can be changed by any function in the program.

```c
#include<stdio.h>

int number;     // global variable

void main()
{
    number = 10;
    printf("I am in main function. My value is %d\n", number);
    fun1();     //function calling, discussed in next topic
    fun2();     //function calling, discussed in next topic
}

/* This is function 1 */
fun1()
{
    number = 20;
    printf("I am in function fun1. My value is %d", number);
}
/* This is function 1 */
fun2()
{
    printf("\nI am in function fun2. My value is %d", number);
}
```

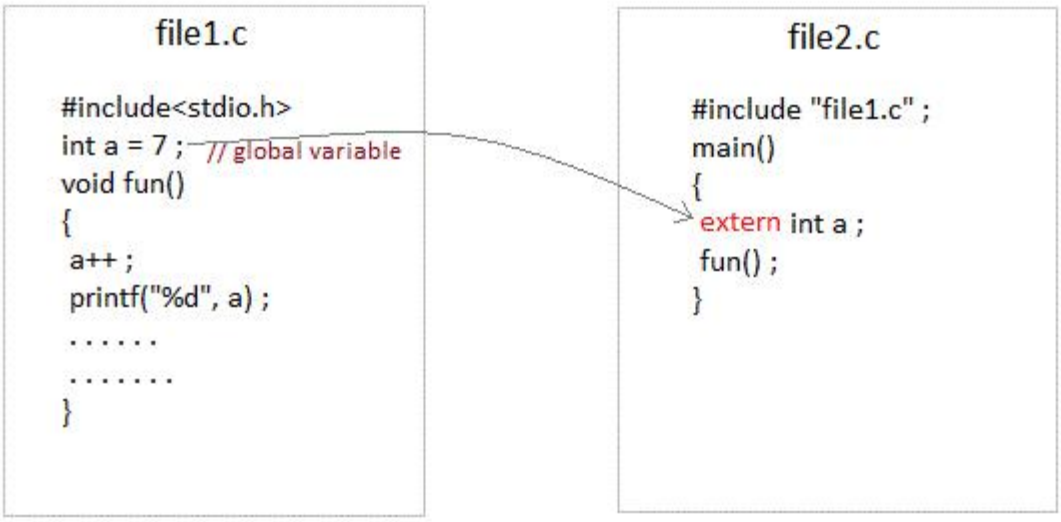
```
I am in function main. My value is 10
I am in function fun1. My value is 20
I am in function fun2. My value is 20
```

Here the global variable `number` is available to all three functions and thus, if one function changes the value of the variable, it gets changed in every function.

Note: Declaring the storage class as global or external for all the variables in a program can waste a lot of memory space because these variables have a lifetime till the end of the program. Thus, variables, which are not needed till the end of the program, will still occupy the memory and thus, memory will be wasted.

`extern` keyword

The `extern` keyword is used with a variable to inform the compiler that this variable is declared somewhere else. The `extern` declaration does not allocate storage for variables.



global variable from one file can be used in other using **extern** keyword.

Problem when extern is not used

```c
int main()
{
    a = 10;     //Error: cannot find definition of variable 'a'
    printf("%d", a);
}
```

Example using extern in same file

```c
int main()
{
    extern int x;   //informs the compiler that it is defined somewhere else
    x = 10;
    printf("%d", x);
}
int x;      //Global variable x
```

Static variables

Scope: Local to the block in which the variable is defined

Default initial value: 0(Zero).

Lifetime: Till the whole program doesn't finish its execution.

A `static` variable tells the compiler to persist/save the variable until the end of program. Instead of creating and destroying a variable every time when it comes into and goes out of scope, `static`variable is initialized only once and remains into existence till the end of the program. A `static`variable can either be internal or external depending upon the place of declaration. Scope of **internal static** variable remains inside the function in which it is defined. **External static** variables remain restricted to scope of file in which they are declared.

They are assigned **0 (zero)** as default value by the compiler.

```
#include<stdio.h>

void test();    //Function declaration (discussed in next topic)

int main()
{
    test();
    test();
    test();
}

void test()
{
    static int a = 0;       //a static variable
    a = a + 1;
    printf("%d\t",a);
}
```

```
1 2 3
```

Register variable

Scope: Local to the function in which it is declared.

Default initial value: Any random value i.e garbage value

Lifetime: Till the end of function/method block, in which the variable is defined.

Register variables inform the compiler to store the variable in CPU register instead of memory. Register variables have faster accessibility than a normal variable. Generally, the frequently used variables are kept in registers. But only a few variables can be placed inside registers. One application of register storage class can be in using loops, where the variable gets used a number of times in the program, in a very short span of time.

NOTE: We can never get the address of such variables.

Syntax :

```
register int number;
```

Note: Even though we have declared the storage class of our variable `number` as register, we cannot surely say that the value of the variable would be stored in a register. This is because the number of registers in a CPU are limited. Also, CPU registers are meant to do a lot of important work. Thus, sometimes they may not be free. In such scenario, the variable works as if its storage class is `auto`.

Which storage class should be used and when

To improve the speed of execution of the program and to carefully use the memory space occupied by the variables, following points should be kept in mind while using storage classes:

- We should use `static` storage class only when we want the value of the variable to remain same every time we call it using different function calls.
- We should use `register` storage class only for those variables that are used in our program very oftenly. CPU registers are limited and thus should be used carefully.
- We should use external or global storage class only for those variables that are being used by almost all the functions in the program.
- If we do not have the purpose of any of the above mentioned storage classes, then we should use the automatic storage class.