Storage Classes in C

Storage Classes are used to describe about 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. This basically signifies that we are not initializing a new variable but instead we are using/accessing the global variable only. The main purpose of using extern variables is that they can be accessed between two different files which are part of a large program. For more information on how extern variables work, have a look at this link.

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 [illegible] are initialized only once and exist till the termination of the program. Thus, no new memory is allocated [illegible]use 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.

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.

To specify the storage class for a variable, the following syntax is to be followed:

Syntax:

```
storage_class var_data_type var_name;
```

Functions follow the same syntax as given above for variables. Have a look at the following C example for further clarification:

```c
// A C program to demonstrate different storage
// classes
#include <stdio.h>

// declaring and initializing an extern variable
extern int x = 9;

// declaring and initialing a global variable z
// simply int z; would have initialized z with
// the default value of a global variable which is 0
int z = 10;

int main()
{
    // declaring an auto variable (simply
    // writing "int a=32;" works as well)
    auto int a = 32;

    // declaring a register variable
    register char b = 'G';

    // telling the compiler that the variable
    // z is an extern variable and has been
    // defined elsewhere (above the main
    // function)
    extern int z;

    printf("Hello World!\n");

    // printing the auto variable 'a'
    printf("\nThis is the value of the auto "
           " integer 'a': %d\n",a);

    // printing the extern variables 'x'
    // and 'z'
    printf("\nThese are the values of the"
           " extern integers 'x' and 'z'"
           " respectively: %d and %d\n", x, z);

    // printing the register variable 'b'
    printf("\nThis is the value of the "
           "register character 'b': %c\n",b);

    // value of extern variable x modified
    x = 2;

    // value of extern variable z modified
    z = 5;

    // printing the modified values of
    // extern variables 'x' and 'z'
    printf("\nThese are the modified values "
           "of the extern integers 'x' and "
           "'z' respectively: %d and %d\n",x,z);

    // using a static variable 'y'
    printf("\n'y' is a static variable and its "
           "value is NOT initialized to 5 after"
           " the first iteration! See for"
           " yourself :)\n");

    while (x > 0)
    {
        static int y = 5;
        y++;

        // printing value of y at each iteration
        printf("The value of y is %d\n",y);
        x--;
    }

    // exiting
    printf("\nBye! See you soon. :)\n");

    return 0;
}
```

Run on IDE

Output:

```
Hello World!

This is the value of the auto  integer 'a': 32

These are the values of the extern integers 'x' and 'z'
respectively: 9 and 10

This is the value of the register character 'b': G

These are the modified values of the extern integers 'x'
and 'z' respectively: 2 and 5

'y' is a static variable and its value is NOT initialized
to 5 after the first iteration! See for yourself :)
The value of y is 6
The value of y is 7

Bye! See you soon. :)
```

Quiz on Storage Classes

This article is contributed by Ayush Jaggi. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : skbarnwal