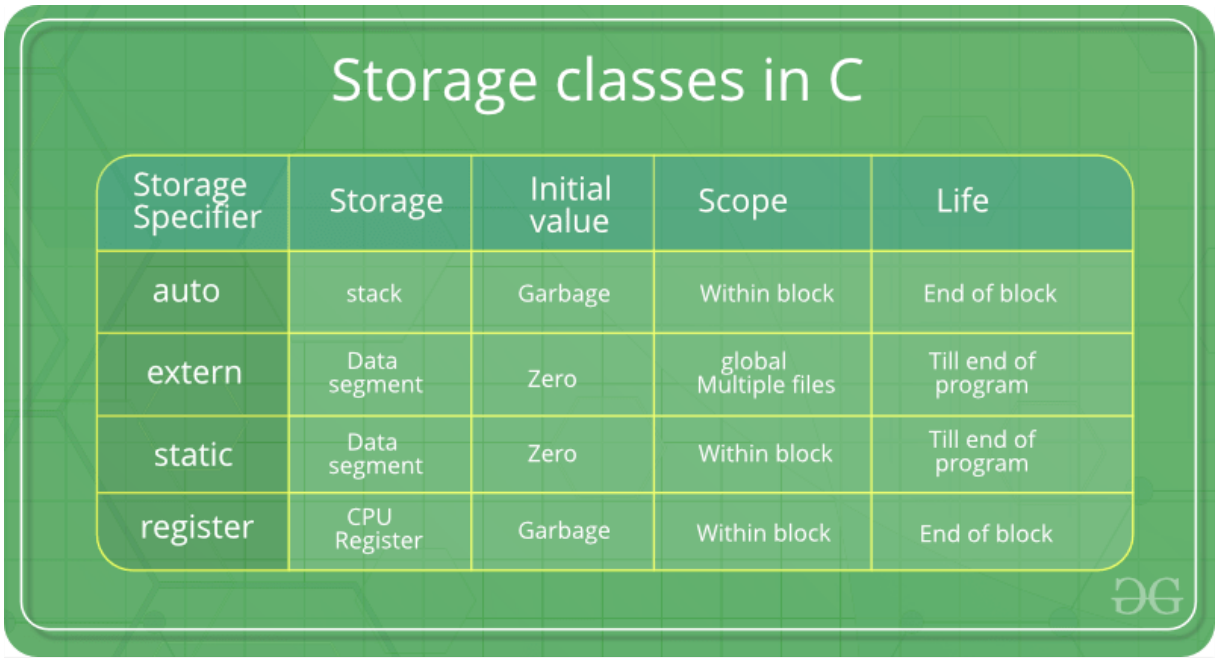
Storage Classes in C

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:



1. **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.

2. **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.

3. **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.

4. **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 the variable which is to be made extern
// an intial value can also be initialized to x
int x;

void autoStorageClass()
{

    printf("\nDemonstrating auto class\n\n");

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

    // printing the auto variable 'a'
    printf("Value of the variable 'a'"
           " declared as auto: %d\n",
           a);

    printf("--------------------------------");
}

void registerStorageClass()
{

    printf("\nDemonstrating register class\n\n");

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

    // printing the register variable 'b'
    printf("Value of the variable 'b'"
           " declared as register: %d\n",
           b);

    printf("--------------------------------");
}

void externStorageClass()
{

    printf("\nDemonstrating extern class\n\n");

    // telling the compiler that the variable
    // z is an extern variable and has been
    // defined elsewhere (above the main
```

```c
    // function)
    extern int x;

    // printing the extern variables 'x'
    printf("Value of the variable 'x'"
           " declared as extern: %d\n",
           x);

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

    // printing the modified values of
    // extern variables 'x'
    printf("Modified value of the variable 'x'"
           " declared as extern: %d\n",
           x);

    printf("---------------------------------");
}

void staticStorageClass()
{
    int i = 0;

    printf("\nDemonstrating static class\n\n");

    // using a static variable 'y'
    printf("Declaring 'y' as static inside the loop.\n"
           "But this declaration will occur only"
           " once as 'y' is static.\n"
           "If not, then every time the value of 'y' "
           "will be the declared value 5"
           " as in the case of variable 'p'\n");

    printf("\nLoop started:\n");

    for (i = 1; i < 5; i++) {

        // Declaring the static variable 'y'
        static int y = 5;

        // Declare a non-static variable 'p'
        int p = 10;

        // Incrementing the value of y and p by 1
        y++;
        p++;

        // printing value of y at each iteration
        printf("\nThe value of 'y', "
               "declared as static, in %d "
               "iteration is %d\n",
               i, y);

        // printing value of p at each iteration
        printf("The value of non-static variable 'p', "
               "in %d iteration is %d\n",
               i, p);
    }

    printf("\nLoop ended:\n");

    printf("---------------------------------");
}

int main()
{

    printf("A program to demonstrate"
           " Storage Classes in C\n\n");

    // To demonstrate auto Storage Class
    autoStorageClass();

    // To demonstrate register Storage Class
    registerStorageClass();
```

```
    // To demonstrate extern Storage Class
    externStorageClass();

    // To demonstrate static Storage Class
    staticStorageClass();

    // exiting
    printf("\n\nStorage Classes demonstrated");

    return 0;
}

// This code is improved by RishabhPrabhu
```

Output:

A program to demonstrate Storage Classes in C

Demonstrating auto class

Value of the variable 'a' declared as auto: 32
——————————
Demonstrating register class

Value of the variable 'b' declared as register: 71
——————————
Demonstrating extern class

Value of the variable 'x' declared as extern: 0
Modified value of the variable 'x' declared as extern: 2
——————————
Demonstrating static class

Declaring 'y' as static inside the loop.
But this declaration will occur only once as 'y' is static.
If not, then every time the value of 'y' will be the declared value 5 as in the case of variable 'p'

Loop started:

The value of 'y', declared as static, in 1 iteration is 6
The value of non-static variable 'p', in 1 iteration is 11

The value of 'y', declared as static, in 2 iteration is 7
The value of non-static variable 'p', in 2 iteration is 11

The value of 'y', declared as static, in 3 iteration is 8
The value of non-static variable 'p', in 3 iteration is 11

The value of 'y', declared as static, in 4 iteration is 9
The value of non-static variable 'p', in 4 iteration is 11

Loop ended:
——————————

Storage Classes demonstrated

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

Recommended Posts:

C | Storage Classes and Type Qualifiers | Question 3
C | Storage Classes and Type Qualifiers | Question 19
C | Storage Classes and Type Qualifiers | Question 19
C | Storage Classes and Type Qualifiers | Question 19
C | Storage Classes and Type Qualifiers | Question 6
C | Storage Classes and Type Qualifiers | Question 7
C | Storage Classes and Type Qualifiers | Question 9
C | Storage Classes and Type Qualifiers | Question 19
C | Storage Classes and Type Qualifiers | Question 11
C | Storage Classes and Type Qualifiers | Question 12
C | Storage Classes and Type Qualifiers | Question 18
C | Storage Classes and Type Qualifiers | Question 17
C | Storage Classes and Type Qualifiers | Question 19
C | Storage Classes and Type Qualifiers | Question 19
C | Storage Classes and Type Qualifiers | Question 14

Improved By : skbarnwal, RishabhPrabhu, harkiran78

Article Tags : C | C Basics | C-Storage Classes and Type Qualifiers | Veritas

Practice Tags : Veritas | C

6

To-do Done

2.7

Based on **39** vote(s)

Feedback/ Suggest Improvement | Add Notes | Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Previous

A C Programming Language Puzzle

Next

Operators in C | Set 1 (Arithmetic Operators)

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

Share this post!

Most popular in C

- Difference between C and C++
- Dividing a Large file into Separate Modules in C/C++, Java and Python
- strrev() function in C
- Difference between scanf() and gets() in C
- Difference between const int*, const int * const, and int const *

More related articles in C

- C program to store Student records as Structures and Sort them by Name
- Code Optimization Technique (logical AND and logical OR)
- Program to Reverse a String using Pointers

Loader in C/C++

Inline function in C

Advertise Here

GeeksforGeeks
A computer science portal for geeks

5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org

COMPANY

About Us
Careers
Privacy Policy
Contact Us

LEARN

Algorithms
Data Structures
Languages
CS Subjects
Video Tutorials

PRACTICE

Company-wise
Topic-wise
Contests
Subjective Questions

CONTRIBUTE

Write an Article
Write Interview Experience
Internships
Videos

@geeksforgeeks, Some rights reserved