

Storage Classes In C Programming:

To fully define a variable one needs to mention not only its 'type' but also its 'storage class'. In other words, not only do all variables have a data type, they also have a "Storage Class". If we don't specify the storage class of a variable in its declaration, the compiler will assume a storage class depending on the context in which the variable is used. Thus variables have certain default storage classes.

There are basically two kinds of locations in a computer where such a value may be kept memory and CPU registers. It is the variable's storage class that determines in which of these two types of locations, the value is stored.

Storage classes in C are used to determine the lifetime, visibility, memory location, and initial value of a variable. There are four types of storage classes in C. "Automatic(auto), External(Extern), Static(static), Register(register)".

Variable's Storage Class Tell Us:

1. Where the variable should be stored.
2. What will be the initial value of the variable, if initial value is not specifically assigned.
3. What is the scope of the variable in which functions the value of the variable would be available.
4. What is the life of the variable how long would the variable exist.

Types Of Storage Classes:

1. Automatic Storage Class (auto)
2. Register Storage Class (register)
3. Static Storage Class (Static)
4. External Storage Class (extern)

Automatic Storage Class (auto):

Storage: Memory

Default Value: An unpredictable value, often called a garbage value

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

Life: Till the control remains within the block in which the variable is defined.

Automatic variables are allocated memory automatically at runtime. The visibility of the automatic variables is limited to the block in which they are defined.

The scope of the automatic variables is limited to the block in which they are defined. The automatic variables are initialized to garbage by default.

The memory assigned to automatic variables gets freed upon exiting from the block. The keyword used for defining automatic variables is auto. Every local variable is automatic in C by default.

The first screenshot shows the source code in `STORAGE.c` and its output in `STORAGE.exe`. The source code defines three automatic variables: `int num1`, `char num2`, and `float num3`. The output shows their values: `Num1 Value : 0`, `Num2 Value :`, and `Num3 Value : 0.000000`. The process exited after 17.2 seconds with return value 0.

The second screenshot shows the same source code with the `auto` keyword added to the variable declarations: `auto int num1`, `auto char num2`, and `auto float num3`. The output is the same, showing the values of the variables.

Static Storage Class (static):

Storage: Memory Default Value: Zero

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

Life: Value of the variable persists between different function calls.

The variables defined as static specifier can hold their value between the multiple function calls. Static local variables are visible only to the function or the block in which they are defined.

A same static variable can be declared many times but can be assigned at only one time. Default initial value of the static integral variable is 0 otherwise null.

The visibility of the static global variable is limited to the file in which it has declared. The keyword used to define static variable is static.

```
STORAGE.c
1 #include<stdio.h>
2 #include<conio.h>
3
4 static char ch;
5 static int in;
6 static float fl;
7
8 int main()
9 {
10     printf("Value Of Ch : %c\n",ch);
11     printf("Value Of Int : %d\n",in);
12     printf("Value Of Fl : %f\n",fl);
13
14     getch();
15 }
16
```

```
E:\C CODE\STORAGE.exe
Value Of Ch :
Value Of Int : 0
Value Of Fl : 0.000000

STORAGE.c
1 #include<stdio.h>
2 #include<conio.h>
3
4 void fun()
5 {
6     static int num1 = 10;
7     auto int num2 = 20;
8
9     printf("%d\t%d\n",num1,num2);
10
11     num1++;
12     num2++;
13 }
14 int main()
15 {
16     for(int i = 0; i < 4; i++)
17     {
18         fun();
19     }
20     getch();
21 }
```

```
E:\C CODE\STORAGE.exe
10      20
11      20
12      20
13      20

-----
Process exited after 52.14 seconds with return value 0
Press any key to continue . . .
```

Register Storage Class (register):

Storage: CPU register Default Value: Garbage Value;

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

Life: Till the control remains within the block in which the variable is defined.

The variables defined as the register is allocated the memory into the CPU registers depending upon the size of the memory remaining in the CPU. We cannot dereference the register variables, i.e., we cannot use &operator for the register variable.

The access time of the register variables is faster than the automatic variables. The initial default value of the register local variables is 0.

The register keyword is used for the variable which should be stored in the CPU register. However, it is compilers choice whether or not; the variables can be stored in the register.

We can store pointers into the register, i.e., a register can store the address of a variable. Static variables cannot be stored into the register since we cannot use more than one storage specifier for the same variable.

```
E:\C CODE\STORAGE.c - [Executing] - Embarcadero Dev-C++ 6.3
File Edit Search View Project Execute Tools AStyle Windows
(globals)
STORAGE.c
1 #include<stdio.h>
2 #include<conio.h>
3
4 int main()
5 {
6     for(register int i = 1; i <= 10; i++)
7     {
8         printf("value : %d\n",i);
9     }
10    getch();
11 }
```

```
E:\C CODE\STORAGE.exe
value : 1
value : 2
value : 3
value : 4
value : 5
value : 6
value : 7
value : 8
value : 9
value : 10
-----
Process exited after 1.188 seconds with return value 0
Press any key to continue . . .
```

```
int main()
{
    register int num = 1000;
    printf("%d\t%u",&num,&num); //showing error
    getch();
}
```

External Storage Class(extern):

Storage: Memory Default Value: Zero

Scope: Global Life: As long as the program's execution doesn't come to an end.

The external storage class is used to tell the compiler that the variable defined as extern is declared with an external linkage elsewhere in the program.

The variables declared as extern are not allocated any memory. It is only declaration and intended to specify that the variable is declared elsewhere in the program.

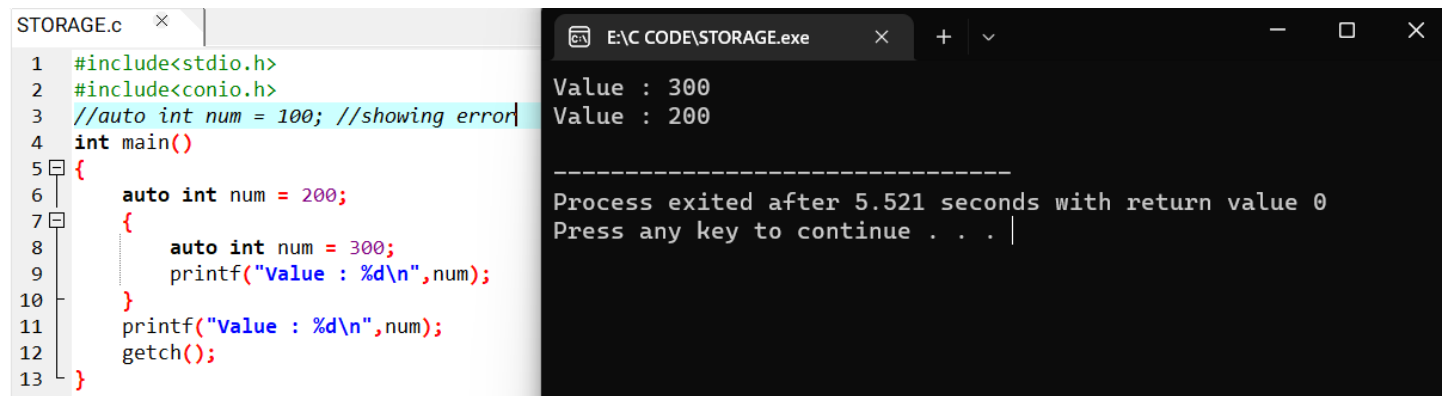
The default initial value of external integral type is 0 otherwise null. We can only initialize the extern variable globally, i.e., we cannot initialize the external variable within any block or method.

An external variable can be declared many times but can be initialized at only once. If a variable is declared as external then the compiler searches for that variable to be initialized somewhere in the program which may be extern or static. If it is not, then the compiler will show an error.

```
STORAGE.c
1 #include<stdio.h>
2 #include<conio.h>
3 int num1 = 100;
4 int num2 = 200;
5 int main()
6 {
7     extern int num1, num2;
8     printf("Value Of Num1 : %d\n",num1);
9     printf("Value Of ++Num1 : %d\n",++num1);
10    printf("Value Of Num2 : %d\n",num2);
11    printf("Value Of ++Num2 : %d\n",++num2);
12    getch();
13 }
```

```
E:\C CODE\STORAGE.exe
Value Of Num1 : 100
Value Of ++Num1 : 101
Value Of Num2 : 200
Value Of ++Num2 : 201
-----
Process exited after 1.698 seconds with return value 0
Press any key to continue . . .
```

Note On Auto Storage Class(auto):



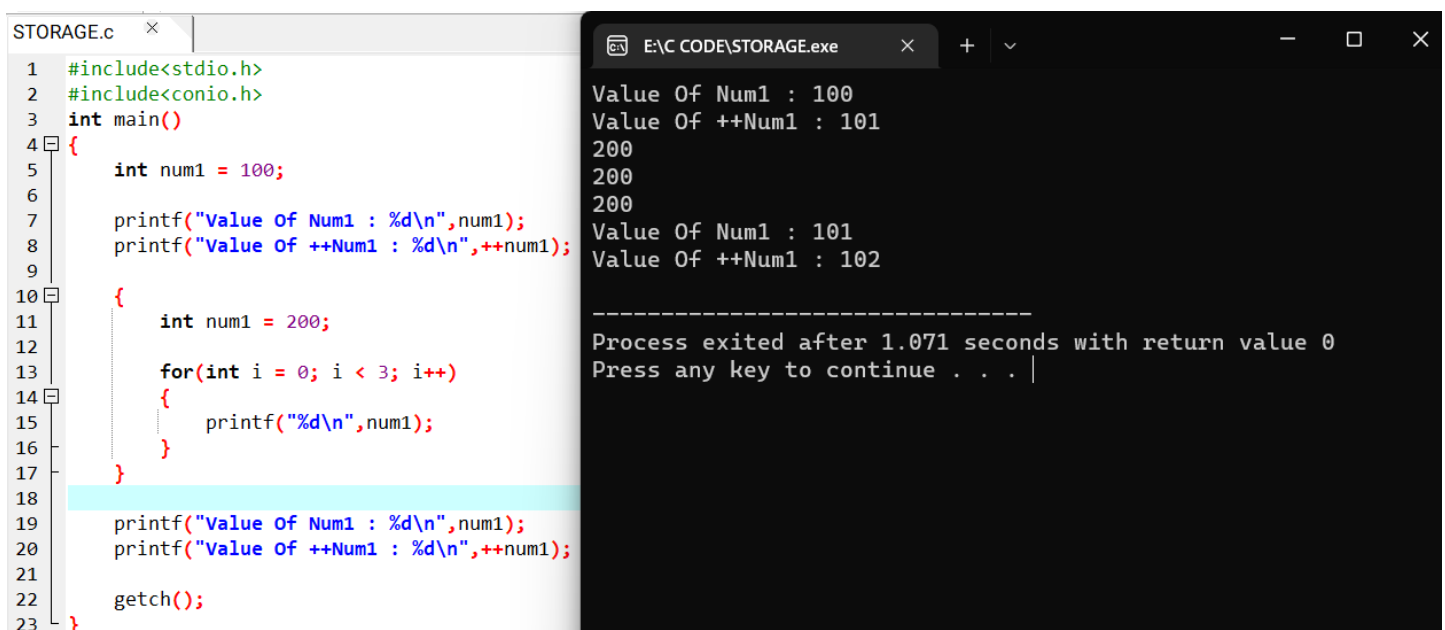
```
STORAGE.c x
1 #include<stdio.h>
2 #include<conio.h>
3 //auto int num = 100; //showing error
4 int main()
5 {
6     auto int num = 200;
7     {
8         auto int num = 300;
9         printf("Value : %d\n",num);
10    }
11    printf("Value : %d\n",num);
12    getch();
13 }
```

```
E:\C CODE\STORAGE.exe x + v
Value : 300
Value : 200

-----
Process exited after 5.521 seconds with return value 0
Press any key to continue . . .
```

Important Note In Storage Class:

1. Use “static” storage class only if you want the value of a variable to persist between different function calls.
2. Use “register” storage class for only those variables that are being used very often in a program. There are very CPU registers at our disposal and many of them might be busy doing something else. Make carefully utilization of the scarce resources. A typical application of register storage class is loop counters, which get used a number of times in a program.
3. Use Extern storage class for only those variables that are being used by almost all the functions in the program. This would avoid unnecessary passing of these variables as arguments when making a function call. Declaring a variables as extern would amount to a lot of wastage of memory space because these variables would remain active throughout the life of the program.
4. If you don’t have any of the express needs mentioned above then use the auto storage class. In fact most of the times we end up using the auto variables. This is because once we have used the variables in a function and are returning from it we don’t mind losing them.



```
STORAGE.c x
1 #include<stdio.h>
2 #include<conio.h>
3 int main()
4 {
5     int num1 = 100;
6
7     printf("Value Of Num1 : %d\n",num1);
8     printf("Value Of ++Num1 : %d\n",++num1);
9
10
11     {
12         int num1 = 200;
13         for(int i = 0; i < 3; i++)
14         {
15             printf("%d\n",num1);
16         }
17     }
18
19     printf("Value Of Num1 : %d\n",num1);
20     printf("Value Of ++Num1 : %d\n",++num1);
21
22     getch();
23 }
```

```
E:\C CODE\STORAGE.exe x + v
Value Of Num1 : 100
Value Of ++Num1 : 101
200
200
200
Value Of Num1 : 101
Value Of ++Num1 : 102

-----
Process exited after 1.071 seconds with return value 0
Press any key to continue . . .
```

Use Of Extern In C Programming In Advance:

```
FILE1.c  ×  FILE2.c  ×
1  #include<stdio.h>
2
3  int count;
4  void extern_function();
5  int main()
6  {
7      extern int num;
8
9      count = 2000;
10
11     extern_function();
12
13     printf("The Value Of Num : %d\n",num);
14     printf("The Value Of Count : %d\n",count);
15 }
16
```

```
FILE1.c  ×  FILE2.c  ×
1  #include<stdio.h>
2
3  int num = 1000;
4
5  void extern_function()
6  {
7      int COUNT = 3000;
8      printf("External File COUNT Value : %d\n",COUNT);
9
10     extern int count;
11     printf("In External File count * 10 Value : %d\n",count*10);
12 }
```

```
E:\C CODE>gcc FILE1.c FILE2.c -o FILE3.exe
```








```
E:\C CODE>FILE3.exe
External File COUNT Value : 3000
In External File count * 10 Value : 20000
The Value Of Num : 1000
The Value Of Count : 2000
```

```
E:\C CODE>gcc -E FILE1.c -o file1.i
```

```
E:\C CODE>gcc -E FILE2.c -o file2.i
```

```
E:\C CODE>gcc -c FILE1.c FILE2.c
```

```
E:\C CODE>
```

| | | | | |
|--|-------------------|---------------|--------|------|
|  FILE1.c | 16-Jan-23 7:50 PM | C Source File | 1 KB | .c |
|  file1.i | 16-Jan-23 8:01 PM | I File | 59 KB | .i |
|  FILE1.o | 16-Jan-23 8:01 PM | O File | 2 KB | .o |
|  FILE2.c | 16-Jan-23 8:00 PM | C Source File | 1 KB | .c |
|  file2.i | 16-Jan-23 8:01 PM | I File | 59 KB | .i |
|  FILE2.o | 16-Jan-23 8:01 PM | O File | 2 KB | .o |
|  FILE3.exe | 16-Jan-23 8:00 PM | Application | 230 KB | .exe |