Storage Classes in C

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
- External
- Static
- Register

Storage Classes	Storage Place	Default Value	Scope	Lifetime
auto	RAM	Garbage Value	Local	Within function
extern	RAM	Zero	Global	Till the end of the main program Maybe d the program
static	RAM	Zero	Local	Till the end of the main program, Ret multiple functions call
register	Register	Garbage Value	Local	Within the function

Automatic

- 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.

Example 1

```
#include <stdio.h>
int main()
{
int a; //auto
char b;
float c;
printf("%d %c %f",a,b,c); // printing initial default value of automatic variables a, b, and c.

return 0;
}
```

Output:

```
garbage garbage garbage
```

Example 2

```
#include <stdio.h>
int main()
{
int a = 10,i;
printf("%d ",++a);
{
int a = 20;
for (i=0;i<3;i++)
{
printf("%d ",a); // 20 will be printed 3 times since it is the local value of a
}
}
printf("%d ",a); // 11 will be printed since the scope of a = 20 is ended.
}
```

Output:

```
11 20 20 20 11
```

Static

- 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.

Example 1

```c
#include<stdio.h>
static char c;
static int i;
static float f;
static char s[100];
void main ()
{
printf("%d %d %f %s",c,i,f); // the initial default value of c, i, and f will be printed.
}
```

Output:

```
0 0 0.000000 (null)
```

Example 2

```c
#include<stdio.h>
void sum()
```

```
{
static int a = 10;
static int b = 24;
printf("%d %d \n",a,b);
a++;
b++;
}
void main()
{
int i;
for(i = 0; i< 3; i++)
{
sum(); // The static variables holds their value between multiple function calls.
}
}
```

Output:

```
10 24
11 25
12 26
```

Register

- 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 can not dereference the register variables, i.e., we can not 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 compiler?s 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 can not be stored into the register since we can not use more than one storage specifier for the same variable.

Example 1

```c
#include <stdio.h>
int main()
{
register int a; // variable a is allocated memory in the CPU register. The initial default value of a is 0.
printf("%d",a);
}
```

Output:

```
0
```

Example 2

```c
#include <stdio.h>
int main()
{
register int a = 0;
printf("%u",&a); // This will give a compile time error since we can not access the address of a register variable.
}
```

Output:

```
main.c:5:5: error: address of register variable ?a? requested
printf("%u",&a);
^~~~~~
```

External

- 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 can not 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.

Example 1

```c
#include <stdio.h>
int main()
{
extern int a;
printf("%d",a);
}
```

Output

```
main.c:(.text+0x6): undefined reference to `a'
collect2: error: ld returned 1 exit status
```

Example 2

```c
#include <stdio.h>
int a;
int main()
{
extern int a; // variable a is defined globally, the memory will not be allocated to a
printf("%d",a);
}
```

Output

```
0
```

Example 3

```c
#include <stdio.h>
int a;
int main()
{
extern int a = 0; // this will show a compiler error since we can not use extern and initializer at same time
printf("%d",a);
}
```

Output

```
compile time error
main.c: In function ?main?:
main.c:5:16: error: ?a? has both ?extern? and initializer
extern int a = 0;
```

Example 4

```c
#include <stdio.h>
int main()
{
extern int a; // Compiler will search here for a variable a defined and initialized somewhere in the pogram or not.
printf("%d",a);
}
int a = 20;
```

Output

```
20
```

Example 5

```c
extern int a;
int a = 10;
#include <stdio.h>
```

```c
int main()
{
printf("%d",a);
}
int a = 20; // compiler will show an error at this line
```

Output

```
compile time error
```