Storage Classes in C

Variable name identifies some physical location within the computer where the variable's value is stored.

There are basically **two kinds of locations in a computer** where such a value may be kept

1. Memory
2. CPU registers.

It is the variable's storage class that determines in which of these two types of locations, the value is stored.

Variable's storage class tells us:

(a) Where the variable would be stored.
(b) What will be the initial value of the variable, if initial value is not specifically assigned. (i.e. the default initial value).
(c) What is the scope of the variable; i.e. in which functions the value of the variable would be available.
(d) What is the life of the variable; i.e. how long would the variable exist.

There are four storage classes in C:
(a) Automatic storage class
(b) Register storage class
(c) Static storage class
(d) External storage class

Automatic Storage Class

Keyword: 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.

//Program on automatic storage class

```
# include <stdio.h>
int main( )
{
auto int i, j ;
printf ( "%d %d\n", i, j ) ;
return 0 ;
}
```

The output of the above program

```
1211 221
```

where,1211 and 221 are **garbage values of i and j.** When you run this program, you may get different values, since garbage values are unpredictable.

//Program on Scope and life of an automatic variable

```
# include <stdio.h>
int main( )
{
auto int i = 1 ;
{
auto int i = 2 ;
{
auto int i = 3 ;
printf ( "%d ", i ) ;
}
printf ( "%d ", i ) ;
}
printf ( "%d\n", i ) ;
return 0 ;
}
```

The output of the above program would be

```
3 2 1
```

Register Storage Class

Keyword: register

Storage: CPU registers.
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.

Note: A value stored in a CPU register can always be accessed faster than the one that is stored in memory. Therefore, if a variable is used at many places in a program, it is better to declare its storage class as register.

//Program on register storge class

```
# include <stdio.h>
int main( )
{
register int i ;
for ( i = 1 ; i <= 10 ; i++ )
printf ( "%d\t", i ) ;
return 0 ;
}
```

The output of the above program would be

```
1	2	3	4	5	6	7	8	9	10
```

Static Storage Class

Keyword: 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.

//Program on static storage class

```
#include <stdio.h>
void increment( ) ;
int main( )
{
increment( ) ;
increment( ) ;
increment( ) ;
return 0 ;
}
void increment( )
{
static int i = 1 ;
printf ( “%d\n”, i ) ;
i = i + 1 ;
}
```

Ouput

```
1
2
3
```

Note: Avoid using static variables Because their values are kept in memory when the variables are not active, which means they take up space in memory that could otherwise be used by other variables.

External Storage Class

Keyword: extern

Storage: Memory.
Default value: Zero.
Scope: Global.
Life: As long as the program’s execution doesn’t come to an end

External variables are declared outside all functions, yet are available to all functions that care to use them.

//Program on external storage class

```
# include <stdio.h>
int i ;
void increment( ) ;
void decrement( ) ;
int main( )
{
printf ( "\ni = %d", i ) ;
increment( ) ;
increment( ) ;

decrement( ) ;
decrement( ) ;
return 0 ;
}
void increment( )
{
i = i + 1 ;
printf ( "on incrementing i = %d\n", i ) ;
}
void decrement( )
{
i = i - 1 ;
printf ( "on decrementing i = %d\n", i ) ;
}
```

output

```
i = 0
on incrementing i = 1
on incrementing i = 2
on decrementing i = 1
on decrementing i = 0
```

//program on external variable show the difference between declaration and definition of a variable

```
# include <stdio.h>
int x = 21 ;
int main( )
{
extern int y ;
printf ( "%d %d\n", x, y ) ;

return 0 ;
}
int y = 31 ;
```

Here, x and y both are global variables

The difference between the following:

There was no need to declare x since its definition is done before its usage . Also remember that a variable can be declared several times but can be defined only once.

```
extern int y ;
int y = 31 ;
```

Here the first statement is a declaration, whereas the second is the definition. When we declare a variable no space is reserved for it, whereas, when we define it space gets reserved for it in memory.

We had to declare y since it is being used in printf( ) before it's definition is encountered.

<u>Difference between auto and static storage class</u>

The difference between them is that

-Static variables don't disappear when the function is no longer active. Their values persist. If the control comes back to the same function again, the static variables have the same values they had last time around.

- if a variable is static, it is initialized to 1 only once. **It is never initialized again.**

// Program on auto storage class

```
#include <stdio.h>
void increment( ) ;
int main( )
{
increment( ) ;
increment( ) ;
increment( ) ;
return 0 ;
}
void increment( )
{
auto int i = 1 ;
printf ( “%d\n”, i ) ;
i = i + 1 ;
}
```

Output

```
1
1
1
```

//program on static storage class

```
#include <stdio.h>
void increment( ) ;
int main( )
{
increment( ) ;
increment( ) ;
increment( ) ;
return 0 ;
}
void increment( )
{
static int i = 1 ;
printf ( “%d\n”, i ) ;
i = i + 1 ;
}
```

Ouput

```
1
2
3
```

Note: In the following statements the first three are definitions, whereas, the last one is a declaration.

```
auto int i ;
static int j ;
register int k ;
extern int l ;
```

Which to Use When

We can make a rules for usage of different storage classes in different programming situations with a view to:
(a) economise the **memory space** consumed by the variables
(b) **improve the speed of execution** of the program

The rules are as under:

Use static storage class, if you want the value of a variable to persist between different function calls.

– **Use register storage class for only those variables that are being used very often in a program.** Reason is, there are very few CPU registers and many of them might be busy doing something else. Make careful 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.

– **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 all the 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.**

– **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 variable. This is because once we have used the variables in a function and are returning from it, we don't mind losing them.