STORAGE CLASSES

INTRODUCTION

From C compiler's point of view, a variable name identifies some physical location within the computer memory, where the string of bits representing the variable's value is stored. 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 locations the value is stored.

The use of the storage classes is that,

- We can have the idea of, the scope of the variable.
- The initial value of the variable, if the initial value is not assigned.
- Where the variable is stored.
- Life time of the variable.

There are four storage classes in C:

1. Automatic storage class
2. Register storage class
3. Static storage class
4. External storage class

Let us examine these storage classes one by one.

TOPIC 1 AUTOMATIC STORAGE CLASS

The variable declared under this storage class has the features like,

- ✓ Storage is in memory.
- ✓ Default initial value is **garbage value**, if initial value is not assigned.
- ✓ The variable life time is, till the control remains within the block in which the variable is defined.

```
main ( )
{
    auto int i, j ;
    printf ( "\n%d %d", i, j ) ;
}
```

The output of the above program could be...

```
1211 221
```

It's better to understand the life and scope of a variable clearly from the following example program.

```
main ( )
{
    auto int i = 1;
    {
        {
            {
                printf ( "\n%d ", i ) ;
                i++;
            }
                printf ( "%d ", i ) ;
        }
```

```
                              printf ( "%d", i ) ; i++;
                    }
                }
```

The output of the above program is: 1 1 1

This is because, all **printf ( )** statements occur within the outermost block (a block is all statements enclosed within a pair of braces) in which **i** has been defined. It means the scope of **i** is local to the block in which it is defined. The moment the control comes out of the block in which the variable store defined, the variable and its value is irretrievably lost.

TOPIC 2 REGISTER STORAGE CLASS

The variable defined under this storage class has the features like,

- ✓ Storage--------------------------- CPU registers.
- ✓ Default initial value ---------------- garbage value.
- ✓ Scope ---------------------------------- local to the block in which it is defined.
- ✓ Life -- till the control remains within the block of the variable which it is defined.

If a value stored in the CPU can be accessed quickly compared to the value stored in memory. At times the variable can be used in the program number of times so in this case for accessing the variable its better to declare the variable under the register storage class.

Note:-

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 at our disposal 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.

Example:-

```
main( )
{
    register int i ;
    for ( i = 1 ; i <= 10 ; i++ )
    printf ( "\n%d", i ) ;
}
```

- This register storage class is used for the faster access of variables.
- Common use in counter.
- Register type declaration is not applicable for arrays,pointers and structures.

TOPIC 3 STATIC STORAGE CLASS

The features of variable defined to have a static storage class are,

- ✓ Storage ---------------------------------memory.
- ✓ Default initial value---------------zero.
- ✓ Scope -------------------------------local to the block in which it is defined.
- ✓ Life ------------------------------------value of the variable persists between two function calls.

Like **auto** variables, **static** variables are also local to the block in which they are declared. 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.

Program -1 :-	Program – 2:-
main()	main()
{	{
Increment ();	Increment ();
Increment ();	Increment ();
}	}
Increment ();	Increment ();
{	{
auto int **I** =1;	static int **I** =1;
printf ("%d",I);	printf ("%d",i);
I = **I** + 1;	**I** = **I** + 1;
}	}
Output :- 1 1	**Output :-** 1 2

In the above example, when variable **I** is **auto**, each time **increment ( )** is called it is re-initialized to one. When the function terminates, **I** vanishes and its new value of 2 is lost. The result: no matter how many times we call **increment ( ), I** is initialized to 1 every time.

On the other hand, if **I** is **static**, it is initialized to 1 only once. It is never initialized again. During the first call to **increment ( ), I** is incremented to 2. Because **I** is static, this value persists. The next time **increment ( )** is called, **I** is not re-initialized to 1; on the contrary its old value 2 is still available. This current value of **I** (i.e. 2) gets printed and then **I = I + 1** adds 1 to **I** to get a value of 3. When **increment ( )** is called the third time, the current value of **i** (i.e. 3) gets printed and once again **i** is incremented. In short, if the storage class is **static** then the statement **static int I = 1** is executed only once, irrespective of how many times the same function is called.

Note:-Use **static** storage class only if you want the value of a variable to persist between different function calls.

TOPIC 4 EXTERNAL STORAGE CLASS

- ✓ Storage: - memory.
- ✓ Life: - as long as the program execution comes to an end.
- ✓ Default value: - zero.
- ✓ Scope: - global.

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

Example:-

```
int i ;

main( )
{
        printf ( "\ni = %d", i ) ;
        increment( ) ;
        increment( ) ;
        decrement( ) ;
        decrement( ) ;
}
increment( )
{
        i = i + 1 ;
        printf ( "\non incrementing i = %d", i ) ;
}
decrement( )
{
        i = i - 1 ;
        printf ( "\non decrementing i = %d", i ) ;
}
```

The output would be:

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

The value of **i** is available to the functions **increment ( )** and **decrement ( )** since **i** has been declared outside all functions.

Note: - The local variable that gets preference over the global variable.

Note:- 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.