Storage Classes

- Storage class is a predefined keyword which specifies where the memory is allocated for different variables.
- A storage class represents the visibility and a location of a variable. It tells from what part of code we can access a variable.

    - Following are the different memory segment of a process:
        - Stack
        - Heap
        - Data Segment
            - BSS (Block started By Symbol)
            - Initialised
        - Text or Code segment

- Following are the storage classes in C:
    - Auto
    - Register
    - Static local and global
    - Extern

1. Auto

a. Auto variables are the one which is declared within the function. Memory for auto local variables will be in the stack segment. Scope of the auto is within the function.

b. Auto stands for automatic storage class. A variable is in auto storage class by default if it is not explicitly specified.

c. Scope of the auto variable is limited within the scope of block only.

d. Once the block completes the execution, the auto variables are destroyed. This means only the block in which the auto variable is declared can access it.

e. For example,

```
Int main()
{
        int num1;
        auto int num2;
}
```

f. Auto variables cannot be declared in the global space.

2. Register

a. Register variable is also a local variable where memory will be in the register.
b. Register variables are used for faster access, because the register is very near to the processor.
c. You can use the register storage class when you want to store local variables within functions or blocks in CPU registers instead of RAM to have quick access to these variables.
d. For example, “delay” or “counter” variables are a good example to be stored in the register.
e. Scope of the variable is within the function.
f. Cannot declare register in the global scope.
g. For example,

```
int main()
{
        register int i;
        for(i = 0; i <1000; i++); //delay loop
}
```

h. Addresses of register variables are inaccessible but a register can hold the address of another variable through a pointer.

3. Static

a. Static variables are declared with the keyword static and memory will be in the data segment as described below:
   - If the static variable is uninitialised then the memory will be in the BSS of the data segment and initialised with 0.
   - If the static is initialised then memory will be in the initialised block of the data segment.
b. Scope of the static variables will be:
   - If declared within the function then scope is within function and life time will be until program execution
   - If declared outside the function then scope is within the file and lifetime is until the program execution.
c. For example,

```
static int gnum; //global static
int main()
{
        static int lnum; //local static
}
```

d. Static variables are best suited when the single instance of the variable is shared in the whole program, like the reservation system; ticket count is the best example for static.

- Extern is used to share the variables from one file to another. Memory will be in the data segment and scope is until program execution.

- Auto and register variables cannot be global, else it will result in compile time error.

- Addresses of the register variables are inaccessible. If you do then it will result in compile time error

- If static variables are global then the scope of the variable will be within the file and cannot be used with extern.

- Extern usage:

File1.c

```
int x = 10;
int main()
{
        foo();
        return 0;
}
```

File2.c

```
int foo()
{
        printf("x in foo %d\n",x);
        return 0;
}
```

- Execute the above program like:
    - gcc file1.c file2.c

    - There will be warning because of missing function prototype, so add function prototype in the file1.c

    - To use x it needs to be externed into that particular file, so in file2.c add extern int x in the global space.

    - Now run the program again. The output will be displayed as
        - X is in foo 10

- Extern always searches for previous visible declarations in the program.

- Last but not the least text or code segment is a read only segment in the memory. Whatever the code is stored in text segments like function code or string literals, those are read only data.