STORAGE CLASSES IN C

STORAGE CLASSES

- To fully define a variable, we need to mention its ***"datatype"*** and its ***"storage class"***.
- A variable's storage class tells us following things about variable :

➢Storage Location

➢Initial Value

➢Scope (which statements can access the value of variable)

➢Lifetime(how long would the variable exist)

Storage classes in C

Storage Specifier	Storage	Initial value	Scope	Life
auto	stack	Garbage	Within block	End of block
extern	Data segment	Zero	global Multiple files	Till end of program
static	Data segment	Zero	Within block	Till end of program
register	CPU Register	Garbage	Within block	End of block

USAGE OF STORAGE CLASSES

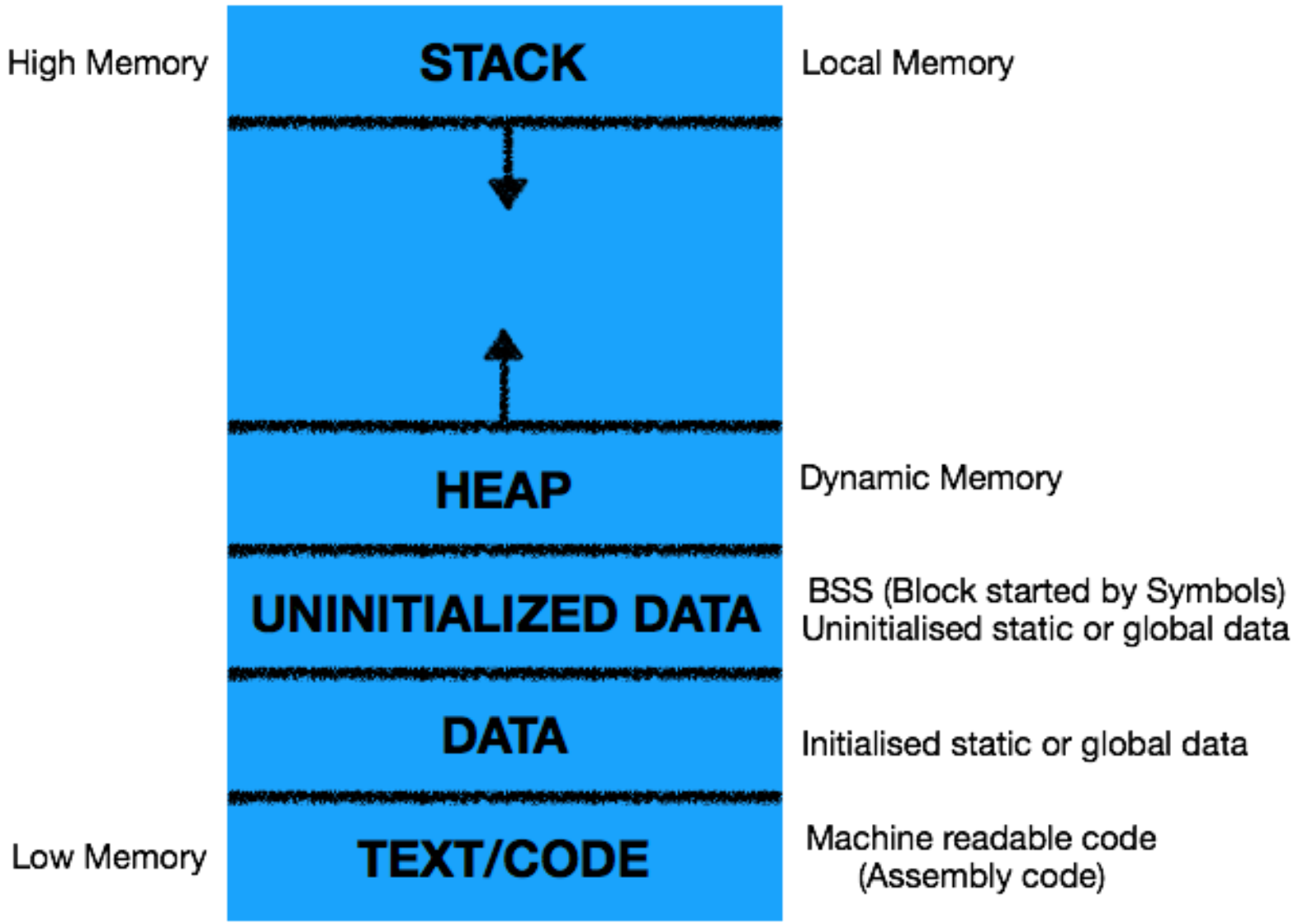
- Use **register** storage class only for those variables that are being used very often in the program, like loop counters.
- Use **static** storage class only if you want the variable to persist across function calls. Common value shared by function calls.
- Use **extern** storage class only for those variables that are being used by almost all the functions in program. This will avoid unnecessary passing of variables as arguments.
- Use **auto** storage class, in absence of any of the needs mentioned above.

extern Variable

- Global or ***extern*** variables have external and internal linkages.
- When you declare a global variable in a file **main.c** before all function definitions, it is visible to all functions in all files within a multi-file program(It has **multi-file scope**). This is known as external linkage.
- If you declare global static variable in **main.c** file, then that variable can be accessed by all functions only in that file (It has **file scope**). It is defined using ***static*** keyword. This is known as internal linkage.

Structure of *.exe* file in Memory

- When ***.exe*** is loaded into memory, it is organized into two areas – CODE segment and DATA segment.
- The CODE or TEXT segment is where the compiled code of the program resides.
- The DATA segment contains the data part of the program
  - STACK section
  - HEAP section
  - DATA section
- These memory segments from top to bottom arranged from low memory to high memory address as shown in the below figure.


High Memory
STACK
Local Memory
HEAP
Dynamic Memory
UNINITIALIZED DATA
BSS (Block started by Symbols)
Uninitialised static or global data
DATA
Initialised static or global data
Low Memory
TEXT/CODE
Machine readable code
(Assembly code)

TEXT/CODE Segment

- TEXT/CODE memory segment stores machine-readable instructions (assembly code).
- Every single line of code in our program converts as its equivalent assembly instruction to execute.
- This segment of memory resides in lowest memory space of the program.

DATA Segment

- This memory segment stores all initialized static or global variables.

- Example:

- int i = 5; // Initialized data, stores in DATA section.

- int main()
- {
  static int =4; // Initialized data, stores in DATA section
- return 0;
- }

Uninitialized DATA Segment

- All un-initialized memory starts after the end of the DATA memory segment.

- int i = 5; // Initialized data, stores in DATA section.
- int j; // Uninitialized data, stores in BSS. Defaults to value 0.
- int main()
- {
- return 0;
- }

HEAP Segment

- This segment of memory starts from the address where BSS ends.
- Heap memory is allocated dynamically using malloc(), calloc() or realloc() functions in C program.

- HEAP memory grows from lower addresses to higher addresses as showing in the above picture.

STACK Segment

- the STACK memory segment starts at the lowest address location and grows towards heap(higher addresses) memory in any program space.

- All auto variables stores in this memory segment, this memory automatically destroys when the scope of the variables expired. Any auto variables inside any function stores into stack address space.

- int i = 5;
- int j;
- int main()
- {
- 	int a = 10;	// Stores in stack.
- 	int b = 5;	// stores in stack
- 	int c = a + b;	// c stores in stack and value is 15.
- 	return 0;
- }
- In this above program, all variables a, b, and c are stores in stack memory. And destroys these memories at the end of main function execution.