STORAGE CLASSES

- A storage class defines the scope (visibility) and life time of variables and/or functions within a C Program.

- There are following storage classes which can be used in a C Program
  - auto
  - register
  - static
  - extern

Auto

Example : { int Count;
auto int Month; }

Storage Class Specifier	auto
Memory	Stack Segment
Default Value	Garbage Value
Scope	Local to the block in which it is defined
Lifetime	Till the end of block
Linkage	None

Register:

Example

```
{
        register int    Count;
}
```

Storage Class Specifier	register
Memory	CPU Register or Stack Segment
Default Value	Garbage Value
Scope	Local to the block in which it is defined
Lifetime	Till the end of block
Linkage	None

Static:

```
static   int   num;
fun(){ static   int   count;  }
```

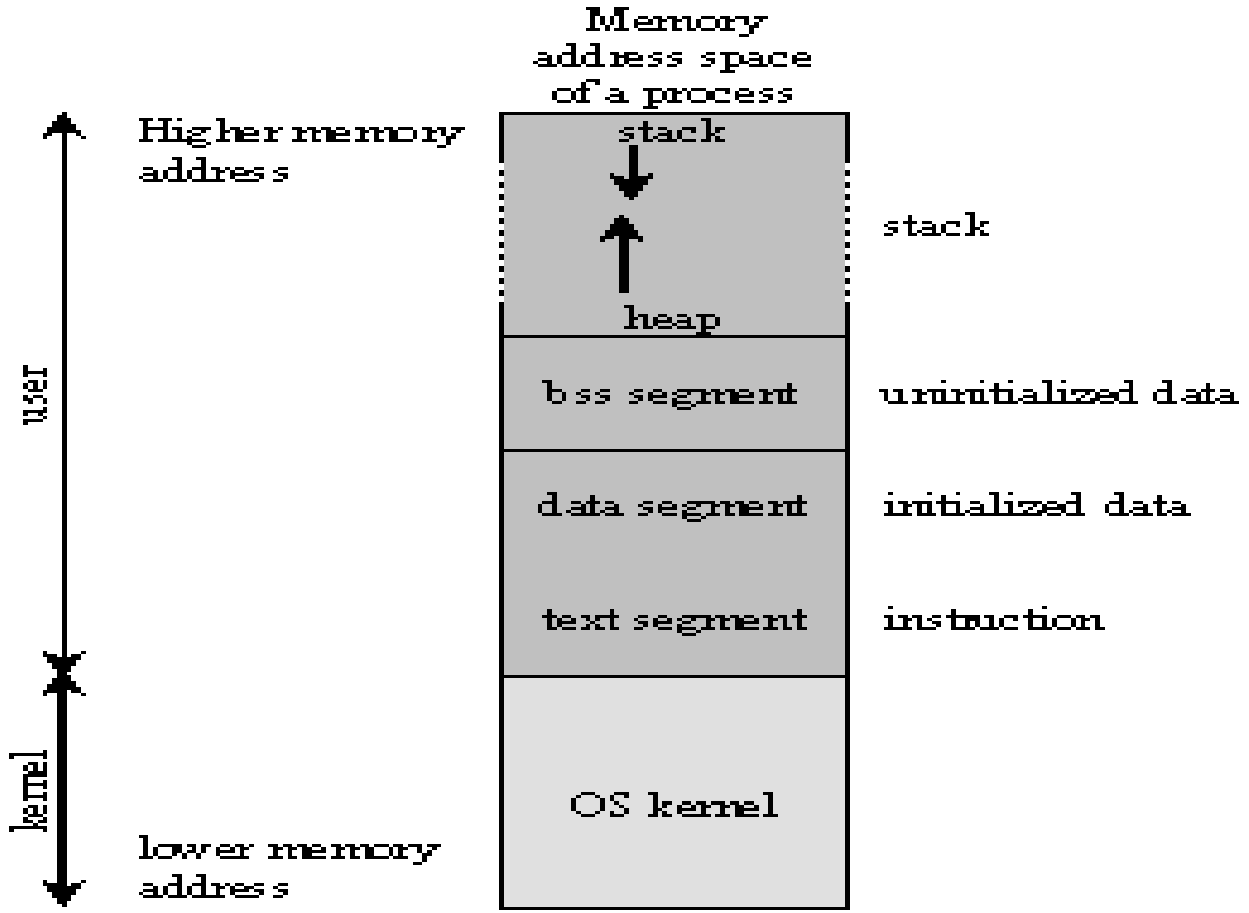
Storage Class Specifier	static
Memory	Data Segment
Default Value	0
Scope	Local to the block in which it is defined
Lifetime	Begin to end of the program
Linkage	Local : None Global : Internal Linkage
Note: Its name is not visible from outside the file in which it is declared	

Extern:

```
extern   int      count;
extern   int      find_sum ( int , int );
```

Storage Class Specifier	extern
Memory	Data Segment
Default Value	Zero
Scope	Throughout the program
Lifetime	Begin to end of program
Linkage	External

Typical Memory Layout of a C program



bss means **B**lock **S**tarted by **S**ymbol

GNU Debugger (GDB)

Starting and invoking gdb

ISM
We inspire you to learn

1. Inserting debugging information inside the output executable files created
after compilation and to start debugging session.

$ gcc -g filename.c

$ gdb ./a.out

2. Giving shell commands from within gdb

(gdb) shell clear

3. Set breakpoint at the function main()

(gdb) break main

4. Delete break point number 1

(gdb) delete 1

Note: Pressing enter with no command executes the previous command

Running and navigating in gdb

1. Run program to be debugged

(gdb) run

2. See where program stopped

(gdb) list

3. Execute next line of the program

(gdb) next (gdb) n

4. Step inside

(gdb) step

5. Print stack trace

(gdb) where

(gdb) frame 0

(gdb) frame 1

6. Return back from function

(gdb) return

7. Continue execution until the next break point.

(gdb) continue

Retrieving values of variables

1. Display the value of a variable "i"

(gdb) display i

2. Set hardware/software watch point for variable "i"

(gdb) watch i

3. Print the value of variable "i"

(gdb) print i

4. Print the address of variable "i"

(gdb) print &i

5. Reassign a value to n

(gdb) set variable n=6

(gdb) continue

6. Call fact() function with different parameters.

(gdb) call fact(4)

7. Display the data type of a variable:

(gdb) ptype i

(gdb) whatis i