**Variable Storage in C**

* global variables -------> data (correct)
* static variables -------> data (correct)
* constant data types -----> code and/or data. Consider string literals for a situation when a constant itself would be stored in the data segment, and references to it would be embedded in the code
* local variables(declared and defined in functions) --------> stack (correct)
* variables declared and defined in main function -----> ~~heap~~ also stack (the teacher was trying to trick you)
* pointers(ex: char \*arr, int \*arr) -------> data or stack, depending on the context. C lets you declare a global or a static pointer, in which case the pointer itself would end up in the data segment.
* dynamically allocated space(using malloc, calloc, realloc) --------> heap

It is worth mentioning that "stack" is officially called "automatic storage class".

**Some heads up:**

1. Whenever a C program is executed some memory is allocated in the RAM for the program execution. This memory is used for storing the frequently executed code (binary data), program variables etc. Below memory segments talks about same:
2. Typically there are 3 type of variables:
   * Local variables (also called as automatic variables in C)
   * Global variables
   * Static variables
   * You can have global static or local static variables but above 3 are parent types.

**5 Memory Segments in C:**

**1. Code Segment**

* Code segment, also referred as text segment, is the area of memory which contains the frequently executed code.
* Code segment is often read-only to avoid risk of getting overridden by programming bugs like buffer-overflow etc.
* Code segment does not contain program variables like local variable (*also called as automatic variables in C*), global variables etc.
* Based on C implementation code segment can also contain read-only string literals. For example, when you do printf("Hello, world") then string "Hello, world" gets created in code/text segment. You can verify this using size command in Linux OS.
* [Further reading](https://en.wikipedia.org/wiki/Code_segment)

**Data Segment**

Data segment is divided in below 2 parts and typically lies below heap area or in some implementations above stack, but data segment never lies between heap and stack area.

2. Uninitialized data segment

* This segment is also known as **bss**.
* This is the portion of memory which contains:
  1. **Uninitialized global variables** ***(including pointer variables)***
  2. **Uninitialized constant global variables**.
  3. **Uninitialized local static variables**.
* Any global or static local variable which is not initialized will be stored in uninitialized data segment
* For example: global variable int globalVar; or static local variable static int localStatic;will be stored in uninitialized data segment.
* If you declare a global variable and initialize it as 0 or NULL then still it would go to uninitialized data segment or bss.
* [Further reading](https://en.wikipedia.org/wiki/.bss)

3. Initialized data segment

* This segment stores:
  1. **Initialized global variables** ***(including pointer variables)***
  2. **Initialized constant global variables**.
  3. **Initialized local static variables**.
* For example: global variable int globalVar = 1; or static local variable static int localStatic = 1; will be stored in initialized data segment.
* This segment can be **further classified into initialized read-only area and initialized read-write area**. *Initialized constant global variables will go in initialized read-only area while variables whose values can be modified at runtime will go in initialized read-write area*.
* ***The size of this segment is determined by the size of the values in the program's source code, and does not change at run time***.
* [Further reading](https://en.wikipedia.org/wiki/Data_segment)

**4. Stack Segment**

* Stack segment is used to store variables which are created inside functions (*function could be main function or user-defined function*), variable like
  1. **Local variables** of the function ***(including pointer variables)***
  2. **Arguments passed to function**
  3. **Return address**
* Variables stored in the stack will be removed as soon as the function execution finishes.
* [Further reading](https://en.wikipedia.org/wiki/Stack-based_memory_allocation)

**5. Heap Segment**

* This segment is to support dynamic memory allocation. If programmer wants to allocate some memory dynamically then in C it is done using malloc, calloc or realloc methods.
* For example, when int\* prt = malloc(sizeof(int) \* 2) then 8 bytes will be allocated in heap and memory address of that location will be returned and stored in ptr variable. ptr variable will be on either stack or data segment depending on the way it is declared/used.
* [Further reading](https://en.wikipedia.org/wiki/Memory_management#HEAP)