In order to understand the difference between stack and heap memory segments, let us first, see how computer's memory is organized for a running program. When a program is loaded into memory, it is organized into two areas of memory called segments: the text segment, and data segment. The text segment (sometimes also called the code segment) is where the compiled code of the program itself resides. The text segment of an executable object file is often read-only segment that prevents a program from being accidentally modified.

In modern operating systems, there is a single text segment, but actually use up to three data segments (data segment, stack segment, and heap segment), depending on the storage class of the data being stored there.

The data segment stores the static data of the program, i.e. the variables that exist throughout program execution. For example, global, static, constant, and external variables (declared with extern keyword).

The stack and heap segments are also memory areas and store program data but there are certain differences between the two on the basis of the type of data they store. Following table presents those differences:

Difference between stack and heap memory segments

Stack	**Heap**
The stack memory segment is RAM's area that is used during program execution for storing return addresses of function calls, arguments to functions, and local variables within functions. Stack memory is maintained in Last In First Out (LIFO) fashion where new storage is allocated and de-allocated at only one end, called the TOP of the stack. A special pointer called stack pointer points to the TOP of the stack. The scope of variables allocated at stack is confined to the block--group of statements enclosed within curly braces. Once the control exits the block the C compiler pops these variables off the stack to clean up.	The heap segment is also RAM's area that is used during program execution but this is used for storing global (storage class external), and static variables. Heap is allocated variables at runtime by usually following 'first fit--choose first block that can satisfy request' strategy. All variables allocated in the heap remains in existence for the duration of a program.
Stack is a limited memory region given to a program for its execution if too much data pushed on to the stack (like large structures and arrays), it can result into a stack overflow and the program will abort. Also, if a pointer is taken to something on the stack, and it is passed to a function or returned it from a function, the function receiving it will result into segmentation fault because the actual data will get popped off and disappear and pointer will be pointing at dead space.	To prevent stack overflow, large memory consuming data items should be allocated on heap on demand and must be freed when they are no longer needed.

Limited stack size is given to a program for its execution and it is OS-dependent.	Size of heap is not limited to a program the way stack is. A program can allocate as much memory dynamically as needed until heap exhausted.
Stack is very fast from access time's point of view.	Heap has slower access time than stack.
Data items on stack are allocated and de-allocated by CPU; therefore memory will be managed efficiently, there will be no memory leaks and memory will not become fragmented.	On the other hand, for the data items allocated on heap, it's programmer's responsibility to allocated memory wisely and free it when it is no longer used by the program. If memory is not well managed there may be memory leaks and memory may become fragmented over time as blocks of memory are allocated.
Variables once allocated on to the stack cannot be resized.	Variables allocated from heap region can be resized using realloc()
Memory allocated from stack is called static memory allocation.	While, memory allocated from heap is called dynamic memory allocation.