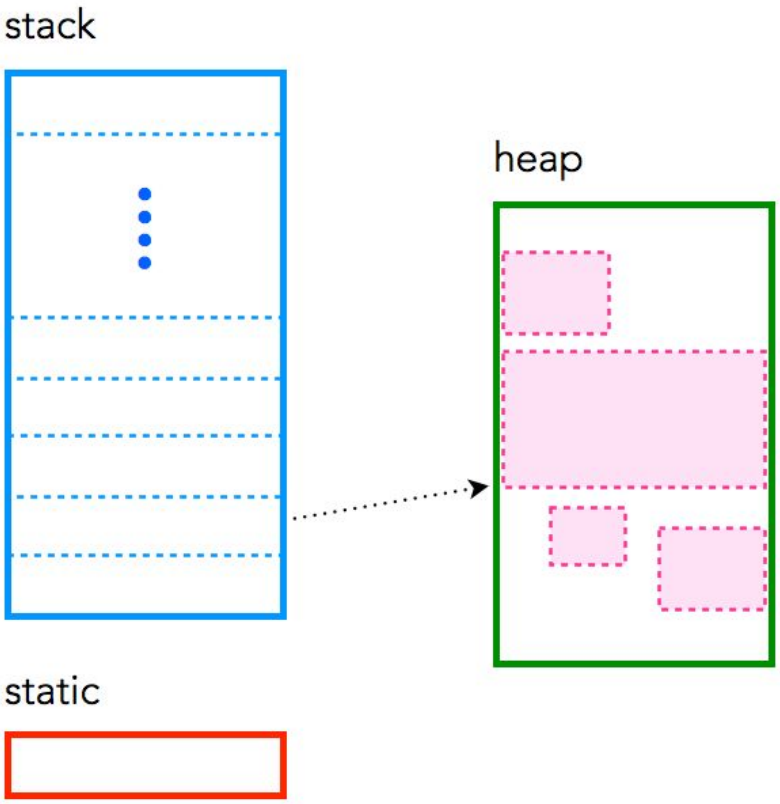
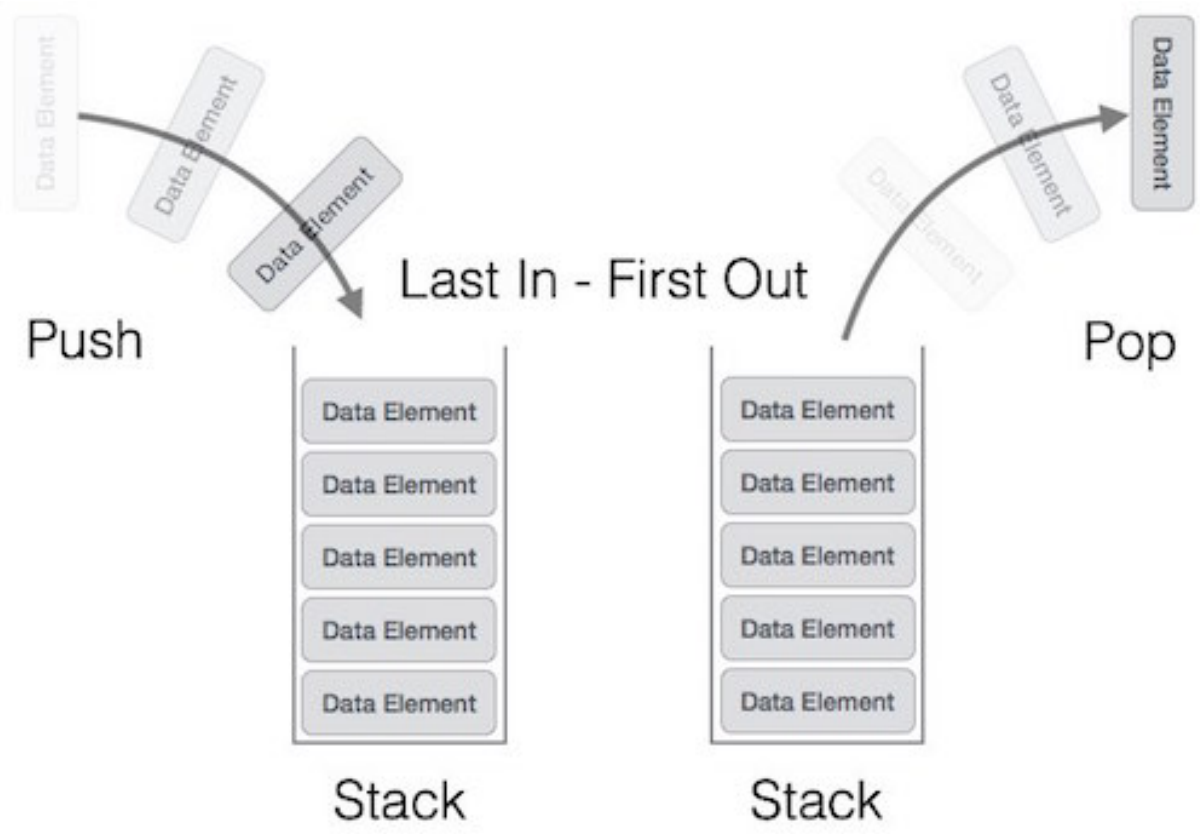
Stack and Heap:

Stack:

- Special place in the memory, where temporary variables live (created by functions)
- LIFO (last in - first out)
- Continuous memory
- Managed, optimized an freed by the CPU -> very fast
- All variables are local and organized in stack frames
- Recursive functions take a whole stack frame for their local variables each time they call themself
- Space is limited (e. g. OSX: 8 MB, stack overflow if you try to put more onto the stack)
- Variables cannot be resized, they exist as long a function is running

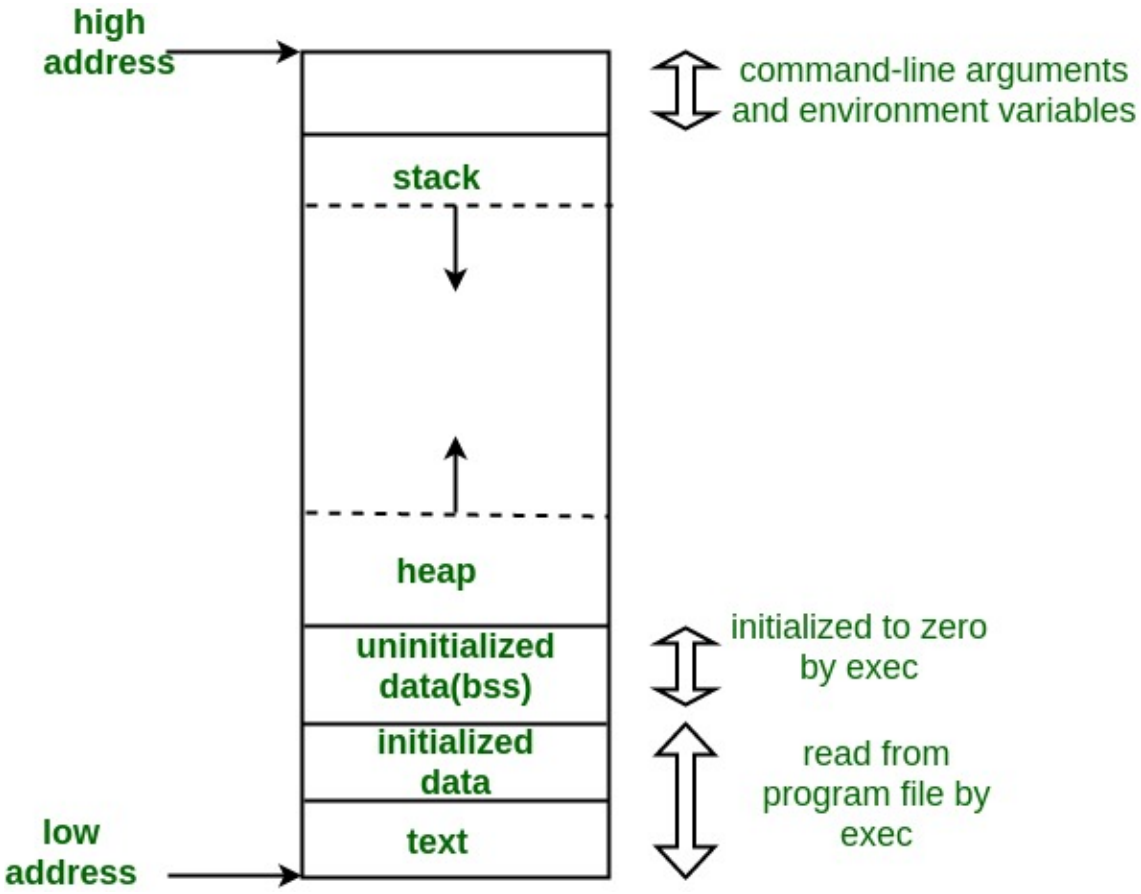


How Lifo works:



Heap:

- Space only physically limited
- No restrictions of variable size (memory size)
- Memory management by the programmer
- Dynamic allocation using C functions `malloc()`, `calloc()` or `realloc()`
- Deallocate memory by yourself using C function `free()`
- Resizing variables using `realloc()`
- Memory leaks possible
- Slightly slower, uses pointers for access
- Useable in global scope
- No continuous memory



Use stack

- dealing with relatively small variables that only need to persist as long as the function using them is alive

Use heap

- allocate a large block of memory (e.g. a large array, or a big struct)
- you need to keep that variable around a long time (like a global)
- you need variables like arrays and structs that can change size dynamically

Hint: don’t try to access a variable that was created on the stack inside some function, from a place in your program outside of that function.

Hint: Don’t use dynamic memory allocation with `malloc()`, `calloc()`, `realloc()` in a perform/real time method (slow)! The needed memory you have to manage outside of these methods.