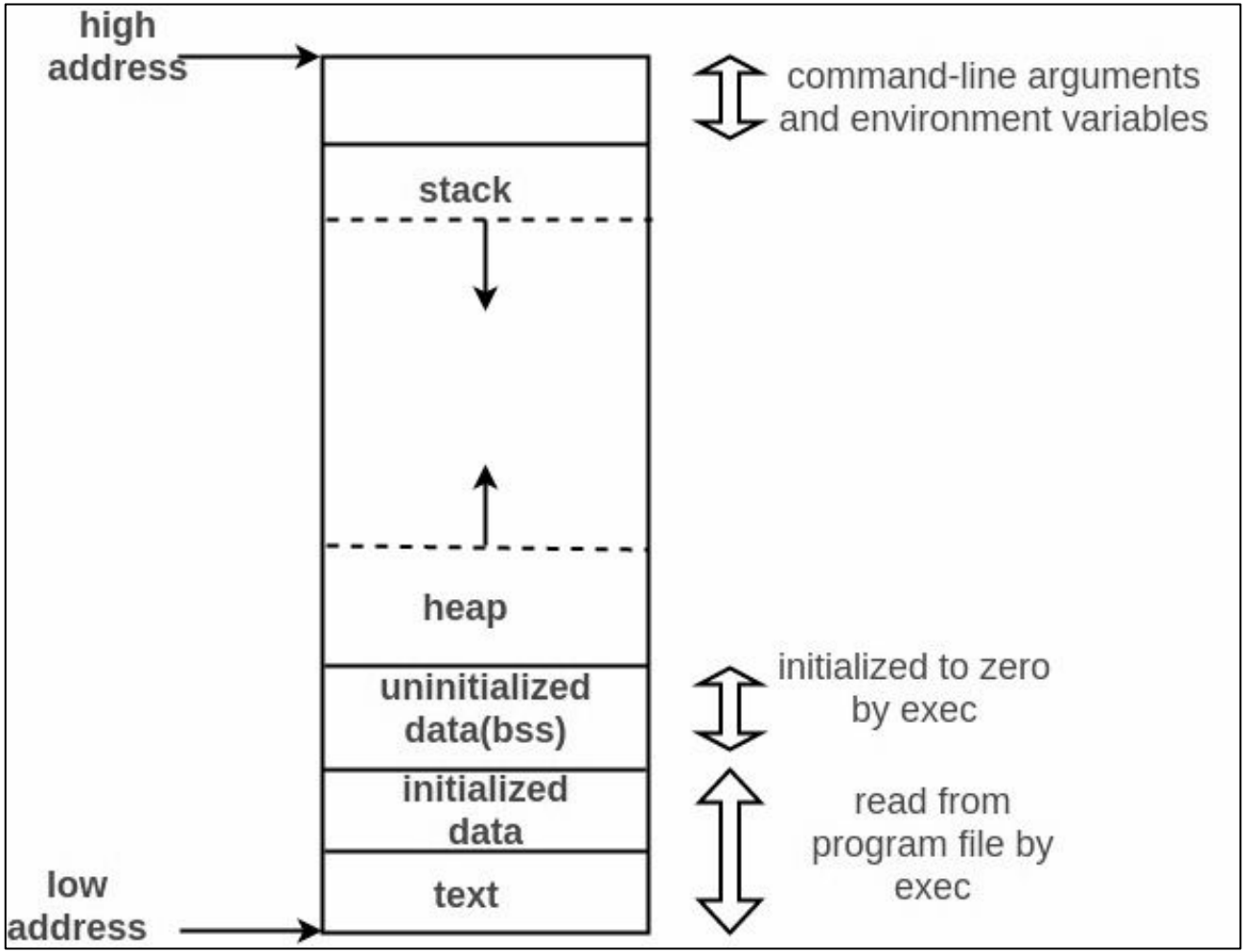
Memory Layout

ADPF

Memory Layout of C Programs

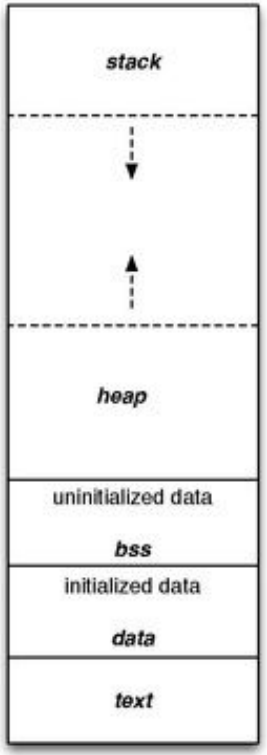
A typical memory representation of C program consists of following sections.

1. Text segment
2. Initialized data segment
3. Uninitialized data segment
4. Stack
5. Heap


high address
command-line arguments and environment variables
stack
heap
uninitialized data(bss)
initialized to zero by exec
initialized data
read from program file by exec
text
low address

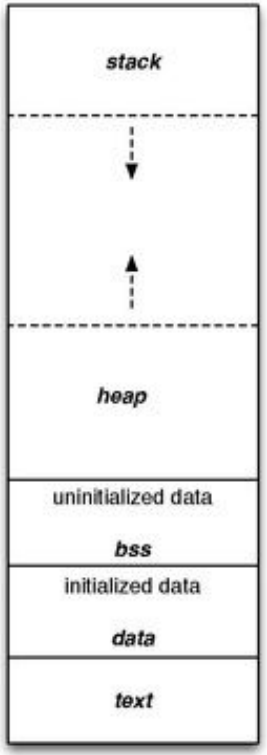
Text (or the code segment)

- This is the area of memory that contains the machine instructions corresponding to the compiled program.
- This area is READ ONLY and is shared by multiple instances of a running program.



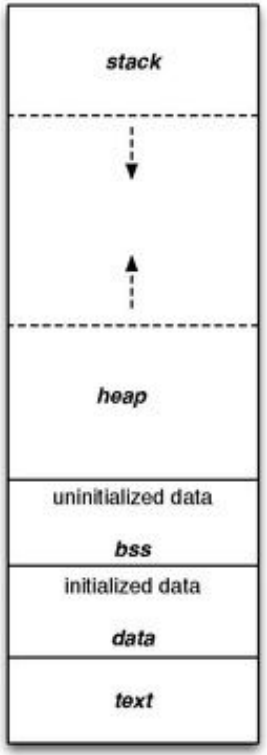
Text (or the code segment)

- A text segment, also known as a code segment or simply as text, which contains executable instructions. [.out file]
- Usually, the text segment is sharable so that only a single copy needs to be in memory for frequently executed programs.
- Also, the text segment is often read-only, to prevent a program from accidentally modifying its instructions.



Data

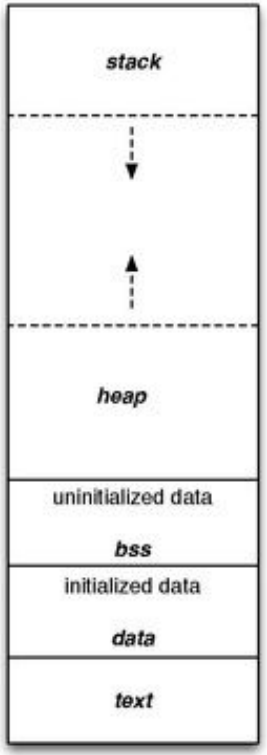
- This area in the memory image of a running program contains storage for initialized global variables.
- This area is separate for each running instance of a program.
- Initialized data is further divided into read-only and read-write section



Data

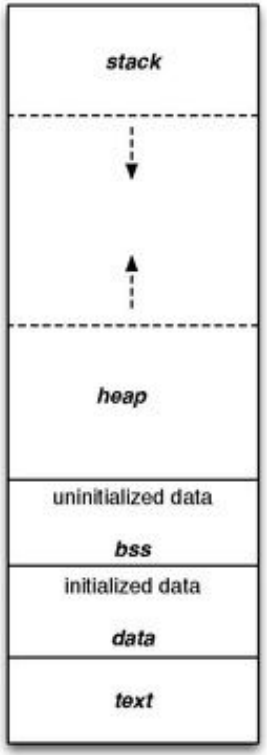
- Initialized data segment, usually called simply the Data Segment.
- A data segment is a portion of the virtual address space of a program, which contains the global variables and static variables that are initialized by the programmer.

```
char a[] = “hello students”
const char* b = “hello DDU”
static int c = 101
int d = 20
```



BSS

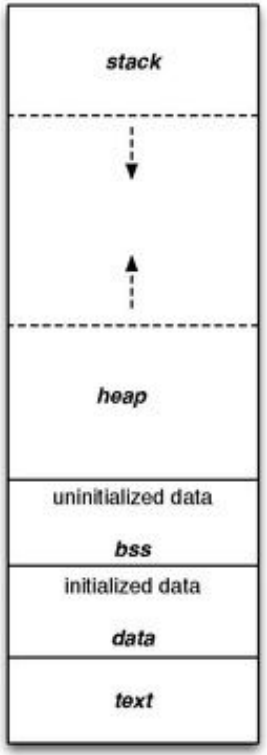
- This is the memory area that contains storage for uninitialized global variables.
- It is also separate for each running instance of a program.



BSS

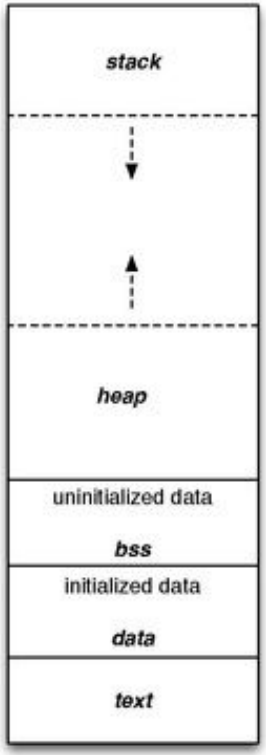
- BSS or Uninitialized data segment [“block started by symbol.”]
- Data in this segment is initialized by the kernel to arithmetic 0 before the program starts executing
- Uninitialized data starts at the end of the data segment and contains all global variables and static variables that are initialized to zero or do not have explicit initialization in source code.

```
static int i
char a[12];
```



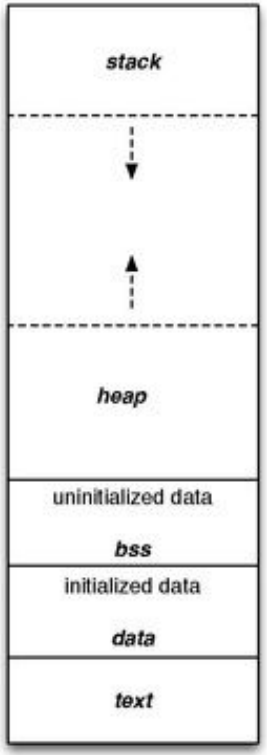
Stack

- This region of the memory image of a running program contains storage for the automatic (local) variables of the program.
- It also stores context-specific information before a function call, e.g., the value of the instruction pointer (program counter) register before a function call is made.
- On most architectures, the stack grows from higher memory to lower memory addresses



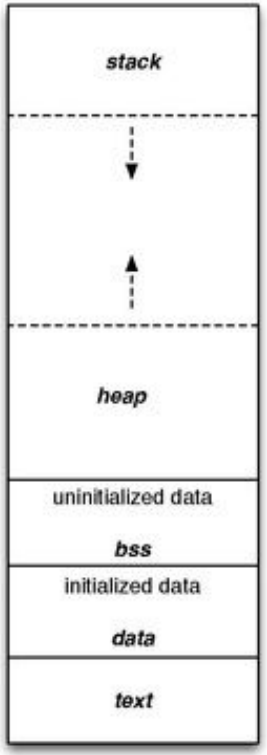
Stack

- The stack area contains the program stack
- LIFO structure
- It is typically located in the higher parts of memory.
- The set of values pushed for one function call is termed a “stack frame”; A stack frame consists at minimum of a return address.



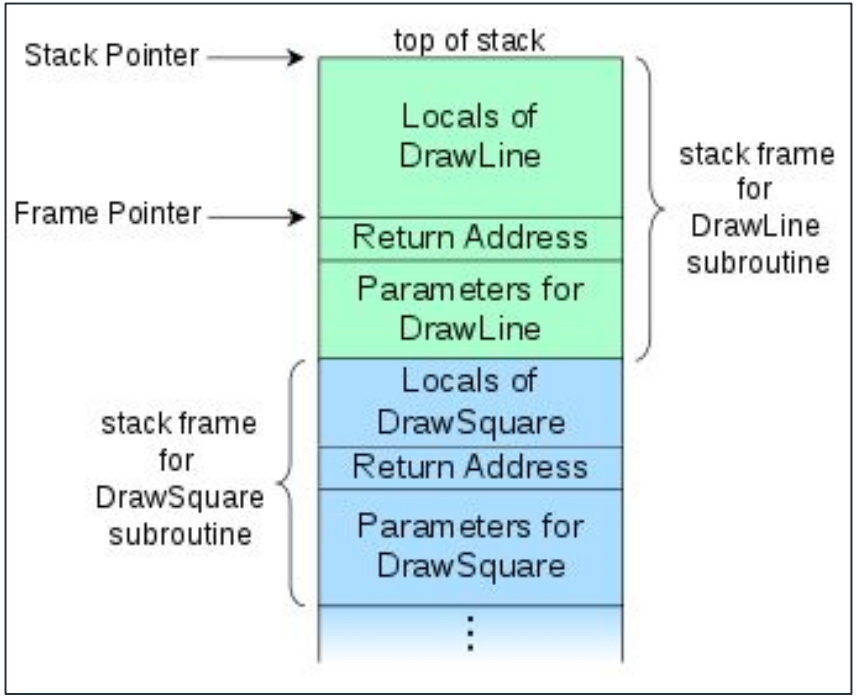
Stack

- In Stack automatic variables are stored, along with information that is saved each time a function is called.
- Each time a function is called, the address of where to return to and certain information about the caller's environment are saved on the stack.
- The newly called function then allocates room on the stack for its automatic and temporary variables.
- This is how recursive functions in C can work.
- Each time a recursive function calls itself, a new stack frame is used, so one set of variables doesn't interfere with the variables from another instance of the function.



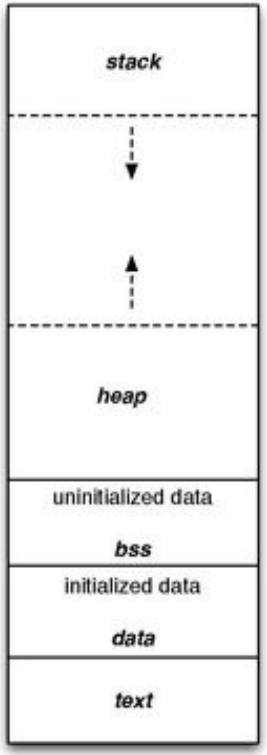
Stack

- For example, if a subroutine DrawSquare calls a subroutine DrawLine from four different places
- DrawLine must know where to return when its execution completes. To accomplish this, the address following the instruction that jumps to DrawLine, the return address, is pushed onto the top of the call stack with each call.



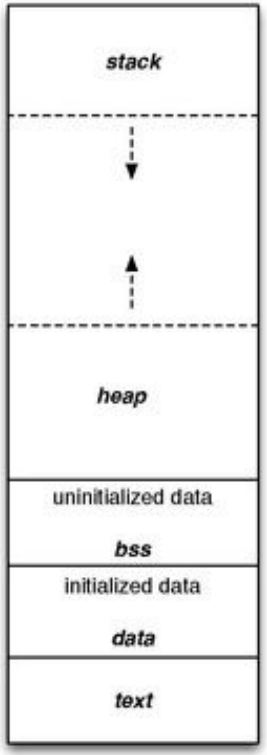
Heap

- This memory region is reserved for dynamically allocating memory for variables at run-time.
- Dynamic memory allocation is done by using the malloc or calloc functions.



Shared libraries

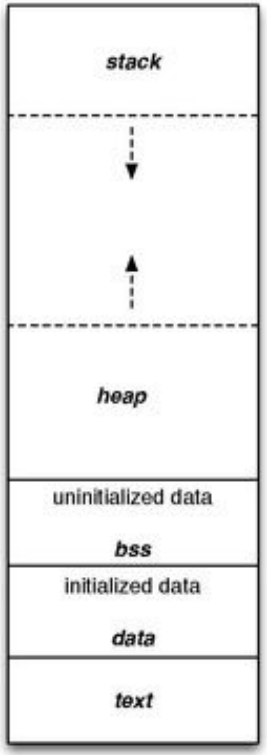
- This region contains the executable image of shared libraries being used by the program.



GCC Command to Verify

- ➔ `gcc PROG.c -o PROG`
- ➔ `size PROG`

text	data	bss	dec	hex	filename
840	264	8	1416	4c0	PROG



Reference

- https://en.wikipedia.org/wiki/Data_segment
- https://en.wikipedia.org/wiki/Call_stack
- https://en.wikipedia.org/wiki/Code_segment
- https://en.wikipedia.org/wiki/.bss

THANK YOU