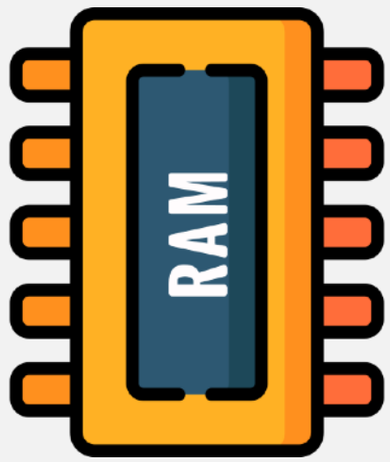
COP 3223H: Introduction to C Programming

Fall 2023

University of Central Florida

Dr. Kevin Moran

Week 12- Class II: Dynamic Memory Allocation - Part 1



Administrivia

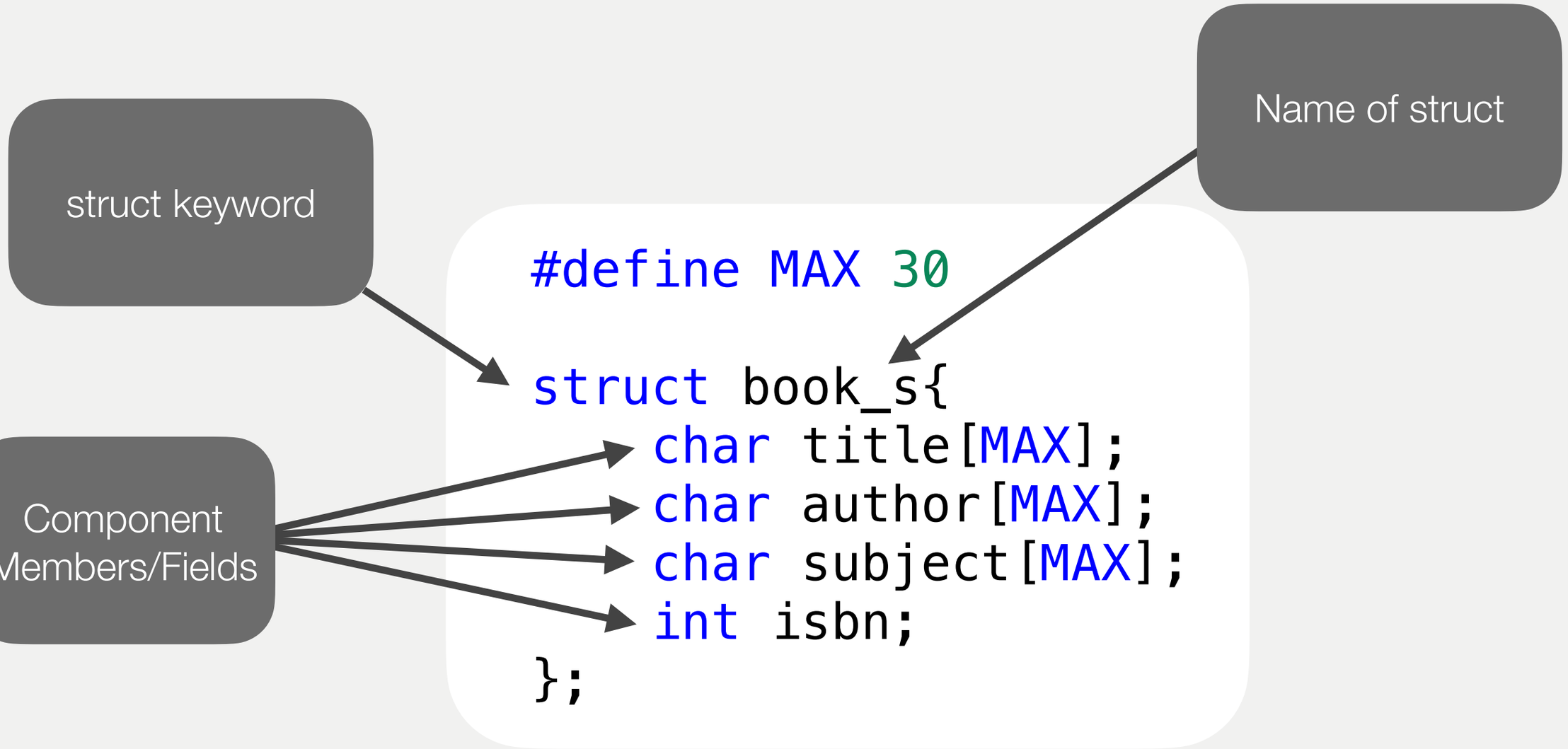
- SPA 3 now due today. - Python script out.
- SPA 4 and LPA 2 have been released, are be due on November, 10th (Friday), and November 17th respectively.
  - SPA 4 Python script coming today.
- Mid-Semester Feedback Survey will be posted today.
  - Please complete to count as a quiz grade.
- No Class on Friday this week (Veterans Day)!
- Office Hours Directly After Class today (visitor to CS Dept.)

Today's Agenda

1. Demo of Structs
2. Intro to Dynamic Memory Allocation in C

Review

User Defined Structure Syntax



Precedence and Associativity of Operators

Precedence	Symbols	Operator	Associativity
Highest	`a[j] f(…)`	Subscripting, function calls, direct component selection	Left
	`++ —`	Postfix increment and decrement	Left
	`++ — ! - + & *`	Prefix increment and decrement, logical not, unary negation and plus, address of, indirection	Right
	`(type name)`	Casts	Right
	`* / %`	Multiplicative operators (multiplication, division, remainder)	Left
	`+ -`	Binary additive operators (addition and subtraction)	Left
	`< > <= >=`	Relational Operators	Left
	`!= =`	Equality/ Inequality Operators	Left
	`&&`	Logical And	Left
	`| |`	Logical Or	Left
Lowest	`+= = -= *= /= %=`	Assignment Operators	Right

Stack Space Visualization with `Structs`

```
#define MAX 30

struct book_s{
    char title[MAX];
    char author[MAX];
    char subject[MAX];
    int isbn;
};

int main(void){

struct book_s mybook;

return 0;

}
```

Stack	Space
AA9	
AA8	
AA7	
AA6	
AA5	
AA4	
AA3	my book.isbn
AA2	mybook.subject
AA1	mybook.author
AA0	mybook, my book.title

Typedef Structures

- As you may of saw, every time we must use struct (such as a declaration), we are required to type out the keyword struct.

- C provides a special keyword that will allow programmers to avoid using the struct keyword.

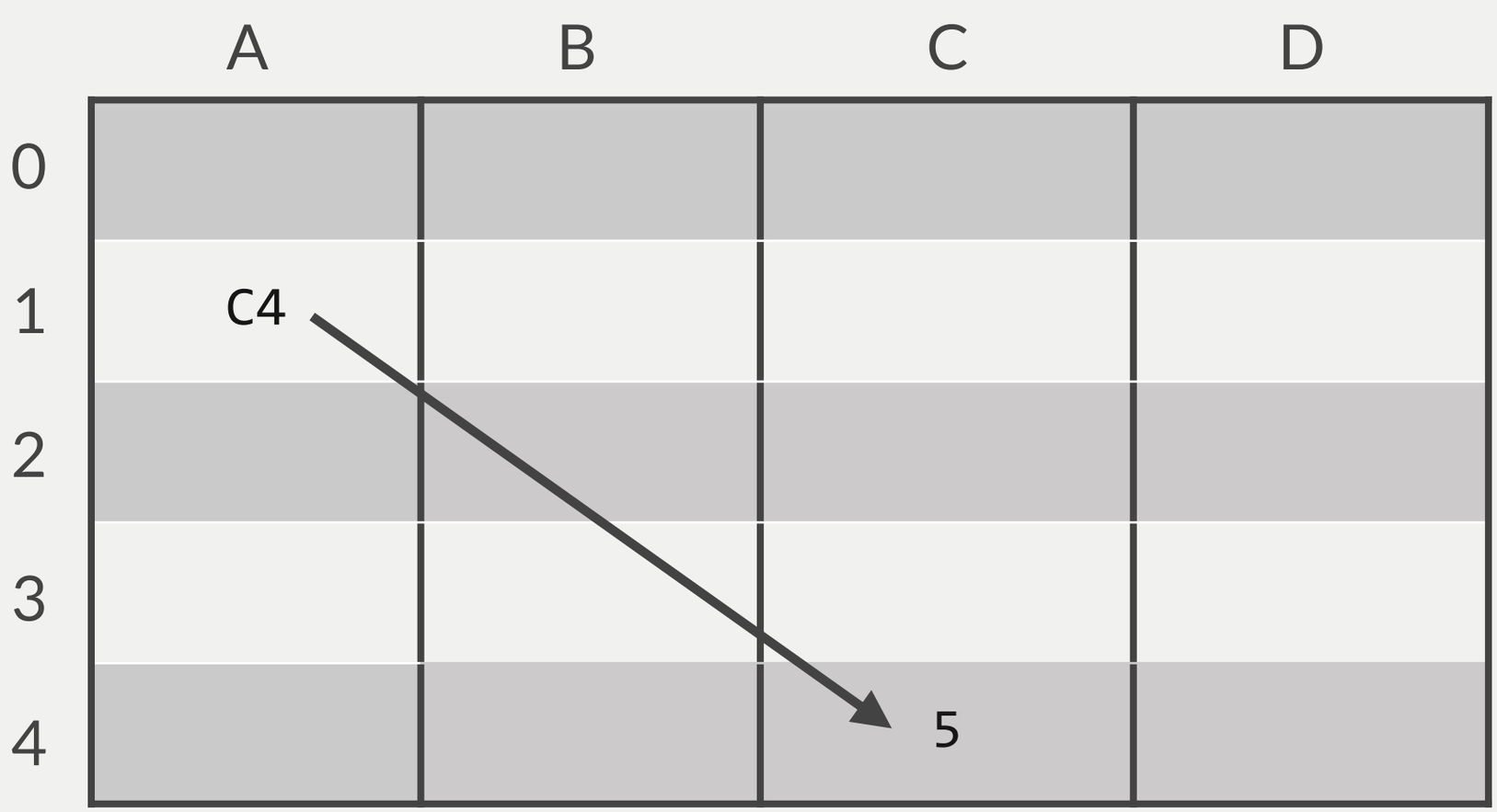
- `typedef` is a special keyword that allows C to assign a name to some type.

```c
#define MAX 30
typedef struct{
    char title[MAX];
    char author[MAX];
    char subject[MAX];
    int isbn;
}book_t;

int main(void){

book_t mybook;

strcpy(mybook.title, "Julius Cesar");
strcpy(mybook.author, "William Shakespeare");
strcpy(mybook.title, "Play");
mybook.isbn = 1234;

return 0;

}
```

Dynamic Memory Allocation

Pointer Refresher

- Special data type that holds an address a memory
- * is the deference operator
- & is the address operator




```c
int x = 5; //C4
int *ptrx = &x; // C4
printf("%d\n", *ptrx); // 5
```

Pointer Uses we have Observed

Use	Implementation
Function Output Parameters	1. Function formal parameter declared a pointer type 2. Actual parameter in a call is the address of the variable
Arrays (strings)	1. Declaration of array variable shows array size 2. Name of array with no subscript is a pointer: meaning the address of the initial array element
File Access	1. Variable declared of type `FILE*` is a pointer to a structure that is to contain access information for a file. 2. File I/O functions such as `fscanf` and `fprintf` expect as arguments file pointers of type FILE *.
Function as a parameter of another function	1. Declaration may or may note include * 2. Name of a function alone (with no parameter list) is a pointer to the function's code

Static Memory

- For this entire course, we have been provided by the OS memory to utilize for our program in the stack space.
  - Limitations:
    - Cannot change the size we are given
      - How can this be potentially bad?
- At compilation time (when code compiles) the memory allocation for the program is predetermined.
- “Get what you get and don’t get upset!”

Dynamic Memory

- Sometimes we may not know how much we really need for a program.
  - Example
    - Array Allocation – what if we allocated 5 elements and realized we need more elements?
- Memory that we can change in size during the program run (different then compilation time).
- Extra memory that we may need during a program is in the ***heap*** space.

`sizeof` Operator

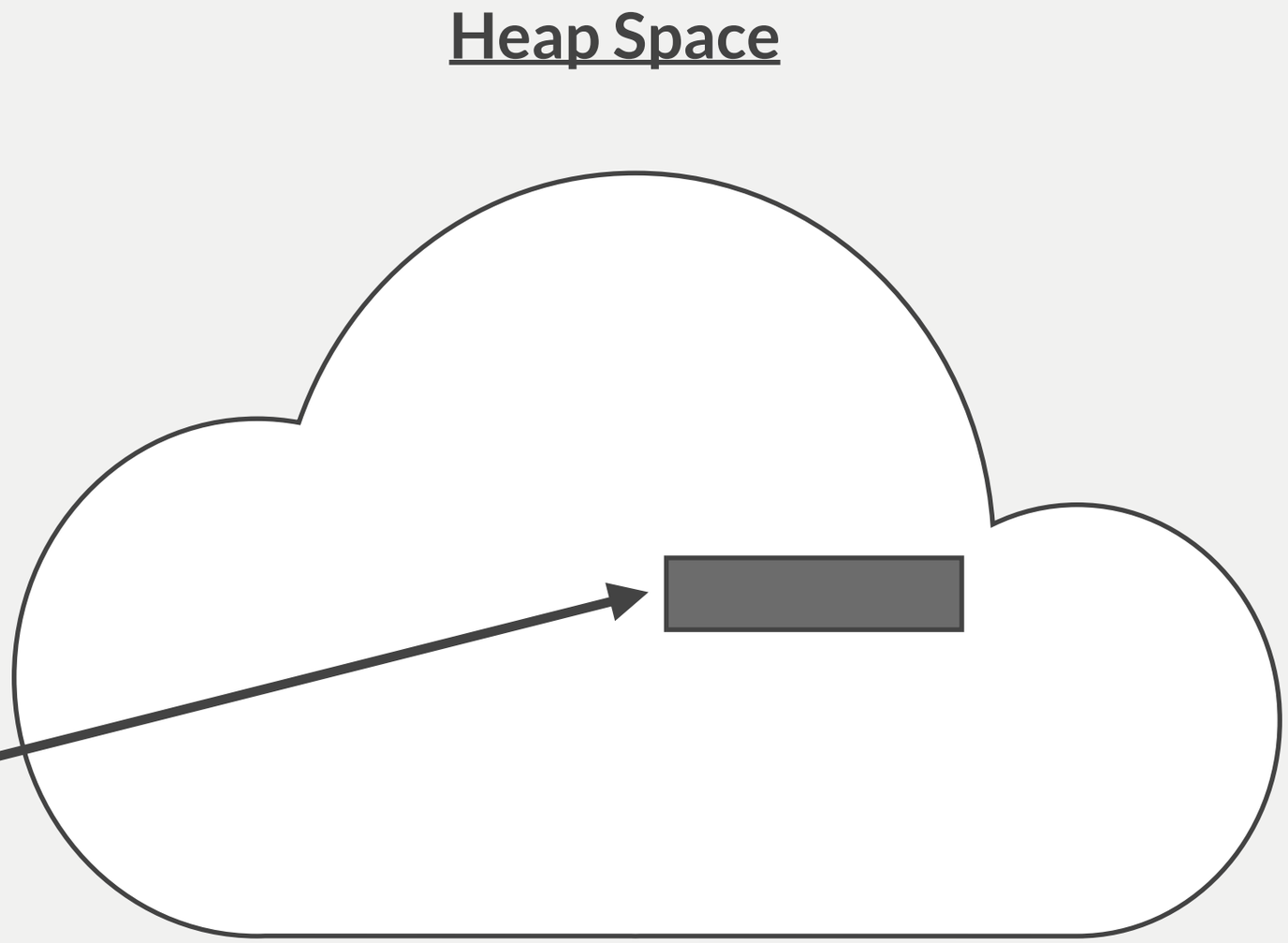
- Returns the size (in bytes) of a data type
  - `sizeof(int)` returns 4 bytes
  - `sizeof(double)` returns 8 bytes
  - `sizeof(char)` returns 1 byte

`malloc()`

- Part of the `stdlib.h` file
- Allocates a single memory block of any built in or user-defined type
- Function that returns memory based on the number of bytes needed
- Parameter of the function takes the number of bytes needed
- The function returns an address or NULL
  - What kind of variable will hold that address?
  - What happens if NULL is returned?
- Heap – region of memory in which the function `malloc` dynamically allocates blocks of storage

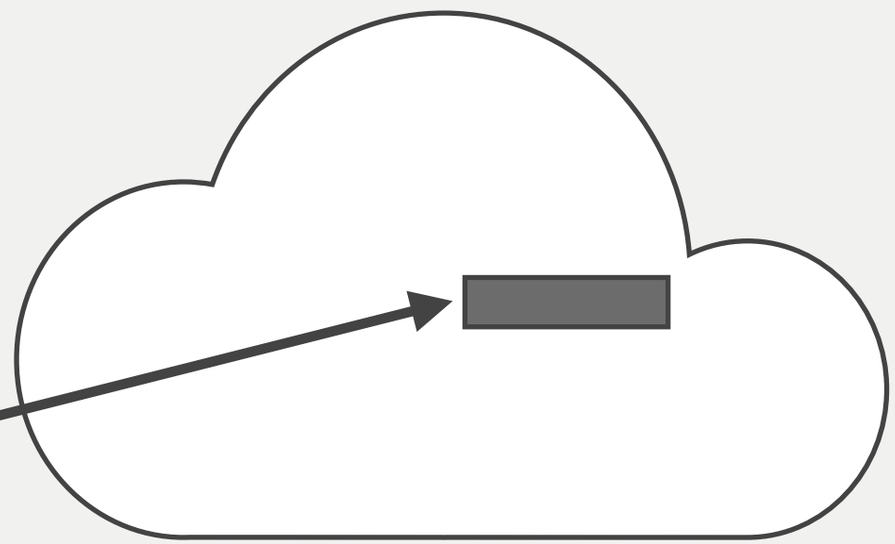
Stack and Heap Space

Stack	Space
AA9	
AA8	
AA7	
AA6	
AA5	
AA4	
AA3	int *ptr
AA2	
AA1	
AA0	

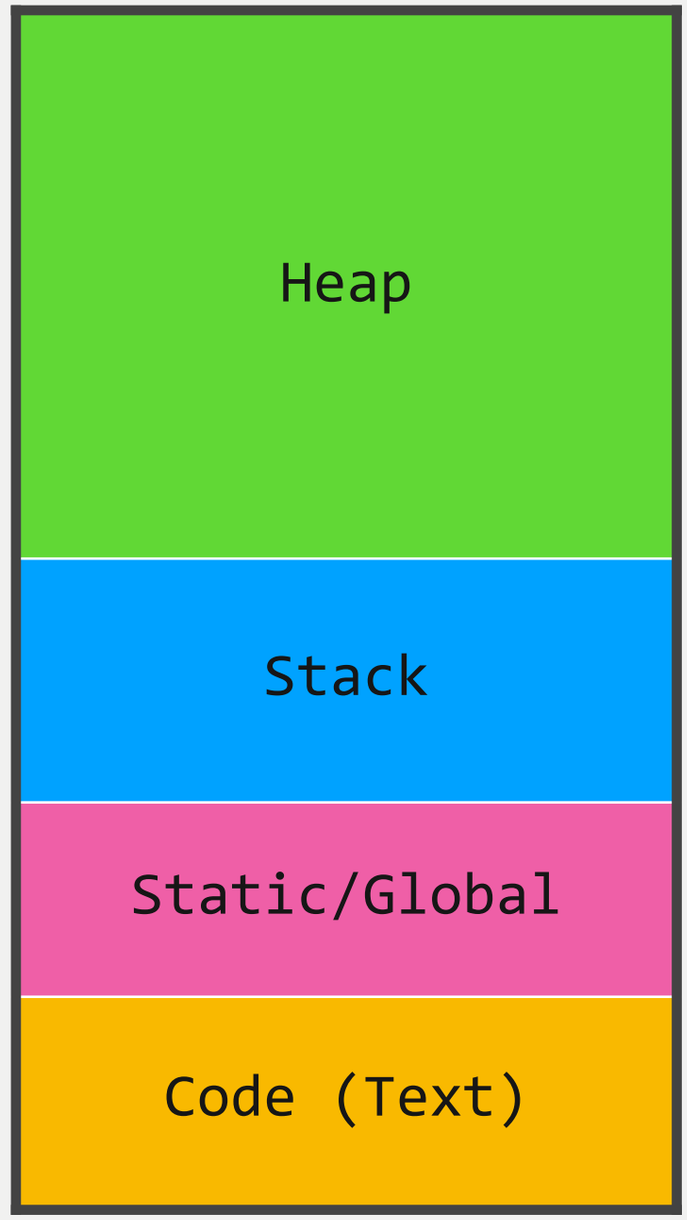


Stack and Heap Space

Stack	Space
AA9	
AA8	
AA7	
AA6	
AA5	
AA4	
AA3	int *ptr
AA2	
AA1	
AA0	



Program
Memory Allocation



`malloc` Example

```
int *pointer;
pointer = malloc(sizeof(int));
```

`free()`

- After we are done with using our dynamic memory we have asked for we need to give it back.
- Why do you think we need to give back memory?
- Parameter is an address in memory (POINTERS!)
- Rule of thumb every `malloc()` call there should be a `free()`.

malloc() + free() Example

```c
int *pointer;
pointer = malloc(sizeof(int));
free(pointer);
```

`free()`

- After **`free()`** is called, the value in the parameter doesn't change.
- Only significant is that the memory is labeled free from the OS perspective
- What do you think this means?
- What should we do with the pointer that is passed in the function call.
  - Set it to NULL!!!

Demo

Acknowledgements

Slides adapted from Dr. Andrew Steinberg's COP 3223H course