COP 3223H: Introduction to C Programming

Fall 2023

University of Central Florida

Dr. Kevin Moran

Class will start in:

10:01

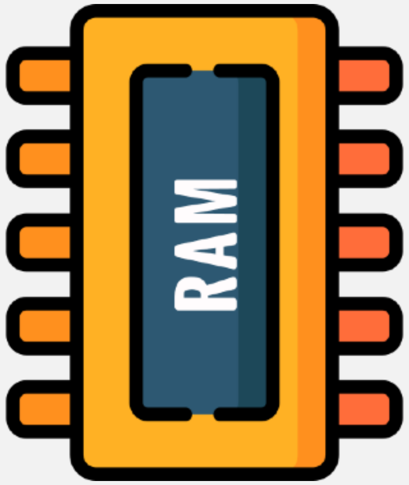
COP 3223H: Introduction to C Programming

Fall 2023

University of Central Florida

Dr. Kevin Moran

Week 13- Class I: Dynamic Arrays



Administrivia

- LPA 2 due on November 17th.
- Mid-Semester Feedback Survey will be posted today.
  - Please complete to count as a quiz grade.
- Office Hours Virtual Today, Dr. Moran still sick.

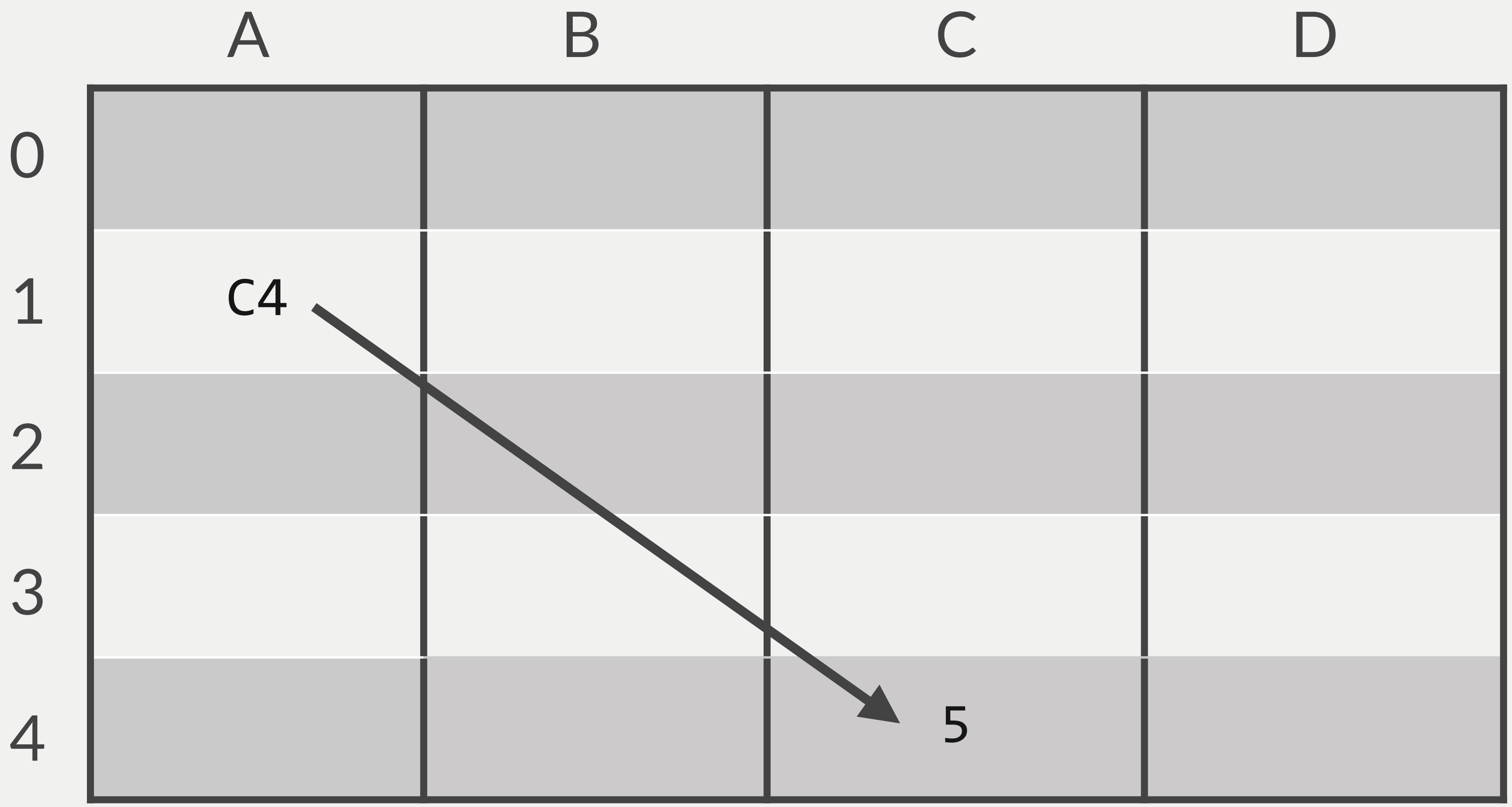
Today's Agenda

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

Review

Pointer Refresher

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



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

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.

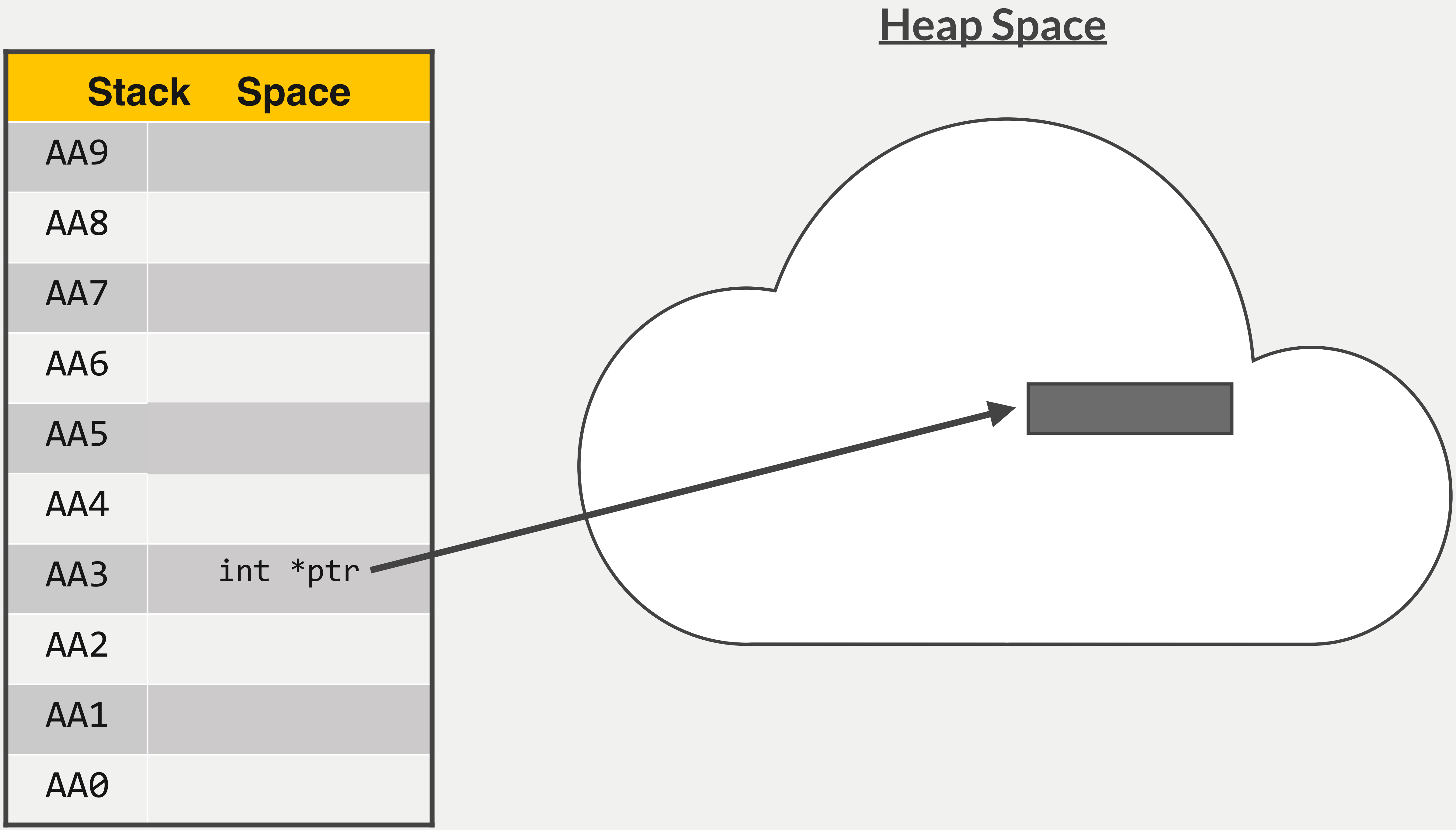
`sizeof` Operator

- 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()`

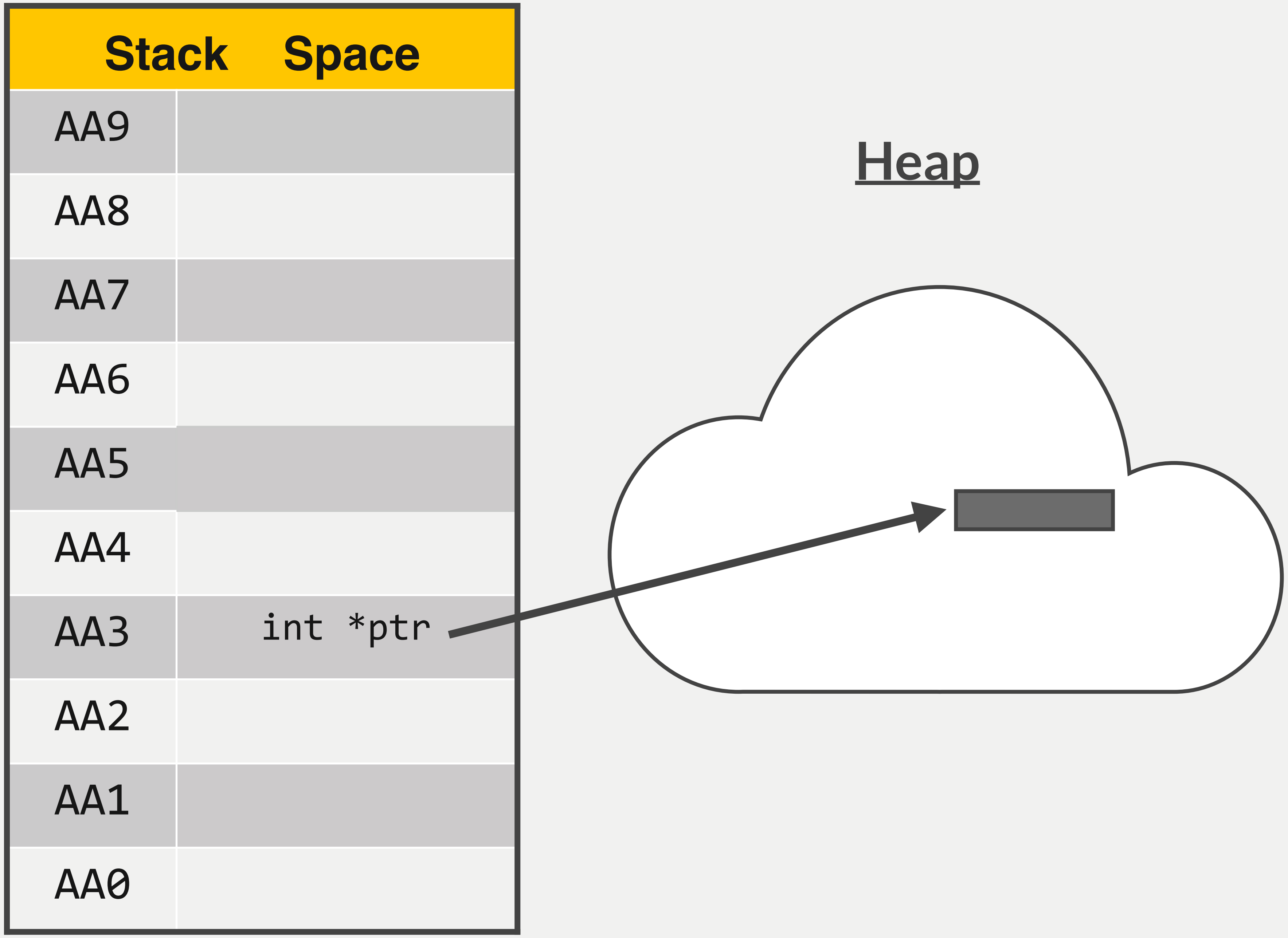
- Part of the `stdlib.h` file
- Allocates a single memory block of any built in or user-defined type
- Function that returns memory based on the number of bytes needed
- Parameter of the function takes the number of bytes needed
- The function returns an address or NULL
  - What kind of variable will 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



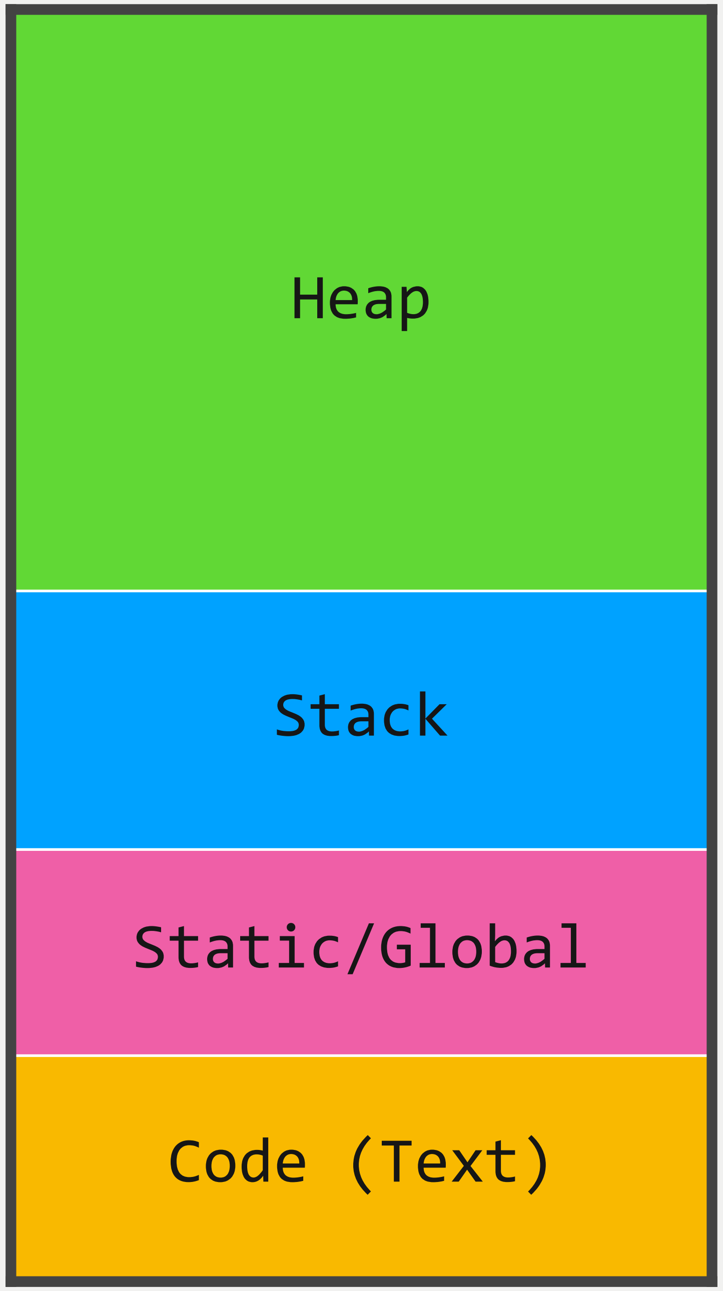
Stack	Space
AA9	
AA8	
AA7	
AA6	
AA5	
AA4	
AA3	int *ptr
AA2	
AA1	
AA0	

Stack and Heap Space



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

Program
Memory Allocation



`malloc` Example

```c
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

```
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

Dynamic Arrays

Variable Length Arrays ☹️

- The arrays we are dealing with use static memory (stack space).
- Static means no flexibility in changing the size of memory required.
- Adding this flexibility results in dynamic memory
- We will study this at the end of the semester.
- Never use variables when declaring an array as you can have potential danger in what the value a variable can hold.
- VLAs pose danger if we accidentally change a value to a size that can’t be properly handled in memory.

Dynamic Array Example

```
int size;

printf ("How many elements would you like: "); scanf(“%d", &size);

int *array = (int *) malloc(size * sizeof(int));

for(int x = 0; x < size; ++x){

    printf("Enter a value: ");
    scanf(“%d", &array[x]) ;
}

for(int x = 0; x < size; ++x){

    printf(“array[%d] = %d\n”, x, array[x]) ;
    free (array) ;
    array = NULL;
}
```

About `sizeof()`

- You have learned that the sizeof operator returns the number of bytes.
- Since dynamic memory returns a heap for a pointer to point at, it will not return the number of elements but instead the size of the pointer.
- So how would you keep track of valid entries in a dynamic array?
  - ***Use a Regular Variable***

Common Functions we can create with Dynamic Arrays

- Insert
- Delete
- Doubling our array
  - If the array is full
- Decrease our array
  - If we are using less than half of the given array
- Search for a value in the array
- Display content
- Sort the data in the array
  - You will learn a lot of sorting techniques in CS1 ☺
- Is the array empty? Meaning there are no valid values stored.

2-D Dynamic Array Example

```
int row, col;

printf("Enter the number of rows and columns you would like. Please separate with a 
space.\n");
printf("Enter here: ");

scanf("%d%d", &row, &col);

int *arr = (int *)malloc(row * col * sizeof(int));

int i,j;
for (i = 0; i < row; i++)
    for(j = 0; j < col; j++)
        *(arr + i*col +j)= i + j;

for (i = 0; i < row; i++){
    for(j=0; j < col; j++){
        printf("&d ", *(arr + i*col + j));
    }
    printf("\n") ;
}
  free(arr) ;
  arr = NULL;
```

Acknowledgements

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