CSE 410 Section 5: Stack vs. Heap

Why Stack Why

Why we can't use int[] as a return type in C.

This stems from the fact that when declaring an array, we must always specify the size of the array. (e.g., int a[10]), but if we set a variable like so: int a[5] = some_func(), where some_func has int[] as a return type, then some_func could return an array of any size.

In theory, this could be alleviated if C added array return types that specified a size (like int[5] some_func()), but this syntax doesn't exist in C. Instead, C requires int* as the return type, and then it's up to the callee of the function to know how many elements are contained in the memory that's pointed to.

What you'll implement.

Write a function int *exp_arr(int base, int size) that returns an array of length size where the i\text{th} index is set to base^i. For example:

```c
int *result = exp_arr(5, 5);
```

result would point to the following array in memory:

```
[1, 5, 25, 125, 625]
```

This array of length 5, and each index corresponds to the result of the parameter 5, as a base with exponent i.

Use wget to grab the starter code from the course website.
https://courses.cs.washington.edu/courses/cse410/18sp/sections/exparr.c

Write the function and test it before reading below.

You may experience an issue... a segmentation fault. A segmentation fault occurs when you attempt to read or write to a piece of memory that you either never allocated or deallocated at some point.

Think for a second why this may have happened BEFORE moving onto the next page (hint: think about the stack).
Our Friend, the Heap

To solve this issue, we introduce the idea of a heap. This is an area in memory which we control as a developer. We choose when to allocate and destroy memory. This memory can be accessed anywhere during execution.

This is very similar to Reference Semantics. In Java, all non-Primitive types are stored on the heap. (Primitives are int, double, short, boolean, etc.) Objects and Arrays are stored on the heap, so when a method in Java updates them, the update is reflected throughout the entire program.

The heap has two main functions we’ll be using: `malloc` and `free`.

Use online resources to figure out how to integrate these into your program, so we get expected output. **Note:** Be sure to include “`#include <stdlib.h>`”, this allows us to use these functions.

Go back and fix the previous function so it executes the expected behavior.

Free that Memory

Test your `exp_arr` code using `valgrind`. `valgrind` is a great tool that checks every time you touch memory to make sure it is valid and that all memory allocated is freed.

Use:

```
valgrind --leak-check=full ./{your_executable}
```

Be sure that `valgrind` reports 0 errors before moving on.

2D Array

Write a function in C that returns a two-dimensional integer array, given the number of rows and columns as parameters.

Use `wget` to grab the starter code from the course website. 
https://courses.cs.washington.edu/courses/cse410/18sp_sections/twodarray.c