LAB #9

Each lab will begin with a brief demonstration by the TAs for the core concepts examined in this lab. As such, this document will not serve to tell you everything the TAs will in the demo. It is highly encouraged that you ask questions and take notes.

In order to get credit for the lab, you need to be checked off by the end of lab. For non-zero lab 8, you can earn full points for completion of the lab outside of lab time, but you must finish the labs before leaving today!!!! For extenuating circumstance, contact your lab TAs and Jennifer Parham-Mocello.

Pair Programming

From this lab forward, you can choose a partner for pair programming. **You must be checked off together. You only need one computer for pair programming.** One person will be the driver, who controls the computer and codes, while the other is the navigator, who observes and advises. After 20 minutes, your TA will switch driver and navigator, continuing this pattern until the task is complete. Please read more about pair programming and the benefits: Pair Programming Student Handout

Recitation Feedback

We would love for you to give us feedback about the peer-led, peer to peer recitations! Please take a few minutes at the beginning of lab to complete this survey:
https://survey.az1.qualtrics.com/jfe/form/SV_8jPq5P2yMAeoIpD
You can find out which recitation section you are in by visiting the Recitation Page.

Statically Allocated 2-D array (5 pts)

First, you can implement this is using a static **2-D array with 3 columns for the 3 posts and 3 rows**, and you can initialize the array with the numbers 1, 2, and 3 in the first column to represent the initial state of the game. The goal is to print out the board after each move in the game, seeing the following output. **Example with two disks**:

```
1  0  0
2  0  0
----------
0  0  0
2  0  1
----------
0  0  0
0  2  1
----------
0  1  0
0  2  0
----------
```

Begin by designing these two functions, **towers() and print_array(). To help you out,** your towers() function will be recursive with the following prototype:

```
void towers(int disks, int b[][3], int from_col, int to_col, int spare);
```

Here is an outline of the recursive towers function:

```
If(number of disks is >= 1)
   Call Towers with (disks-1, b, from_col, spare, to_col)
   Move the disk
   Print the board
   Call Towers with (disks-1, b, spare, to_col, from_col)
```

Dynamically Allocated 2-D array (5 pts)

Next, implement this is using a dynamically allocated **2-D array with 3 columns for the 3 posts and N rows for N disks.** Get the number of disks from the user as a command-line argument, i.e. towers 5.

Continue to initialize the array with the numbers corresponding to the disks in the first column and 0s in all other columns to represent the initial state of the game. You should now see the above example output, given 2 for the number of disks.

Remember to change your towers() and print_array() function parameters to accept dynamically allocated arrays, rather than statically allocated. To help you out, your towers() function will be change to the following prototype:

```
void towers(int disks, int **b, int from_col, int to_col, int spare);
```

Make sure you delete your board after calling the towers function.

Create/Delete Functions for Dynamically Allocated 2-D array

If you haven't done so already, create functions for creating and deleting the array on the heap. Make sure you set the board back to null in the delete function!

Run your program through valgrind to make sure you do not have any memory leaks!!!