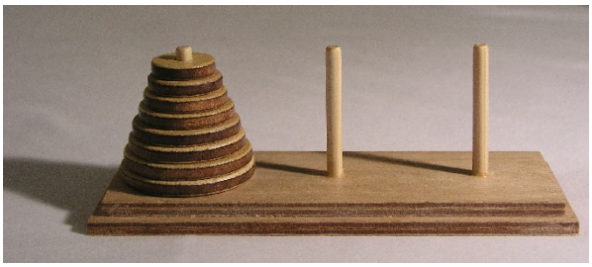
Lab 9

In order to get credit for this lab, you need to be checked off by the end of the period. You can earn a maximum of 3 points for lab work completed outside of lab time, but you must finish the work before the start of your next lab. For extenuating circumstances, contact your lab TAs and Instructor.

Towers of Hanoi is a simple puzzle that we're going to be using as practice to combine the concepts of recursion and 2D arrays. The puzzle itself is very simple– it consists of three pegs arranged from left to right, and some number of disks N of different sizes. To begin, the N disks are placed on the leftmost peg in order of their size, with the largest disk at the bottom of the peg. The puzzle's goal is to finish with the disks arranged in the same order (biggest on the bottom, smallest on the top) on the rightmost peg. Of course, you can't just move the disks however you want! You can only move one disk at a time by taking it off the top of its peg and putting it onto another peg. Additionally, you're not allowed to place a disk on top of another disk that's smaller; that is, every disk must be smaller than every disk beneath it on the peg.



Towers of Hanoi puzzle with N = 8 disks (Wikipedia)

Statically Allocated 2-D array (5 pts)

First, implement this puzzle using a statically allocated 2-D array with 3 columns and 3 rows. 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 columns of the array represent the posts in the game. The goal is to print out the board after each move in the game, seeing the output as shown below.

Example output from recursive solution:

```
1  0  0
2  0  0
3  0  0
----------
0  0  0
2  0  0
3  0  1
----------
0  0  0
0  0  0
3  2  1
----------
0  0  0
0  1  0
3  2  0
----------
0  0  0
0  1  0
0  2  3
----------
0  0  0
0  0  0
1  2  3
----------
0  0  0
0  0  2
1  0  3
----------
0  0  1
0  0  2
0  0  3
----------
```

Begin by designing these two functions, **towers() and print_array().** To help you out, your towers() function will be recursive with the following prototype. Note that the statically allocated implementation can use the [][3] notation in the function prototype since you know that each subarray will have 3 indices. This does not work for a dynamic array when you don't know the number of indices at compile time.

```
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 still see the above example output, given 3 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 verify that you do not have any memory leaks!!!