Towers of Hanoi

*Towers of Hanoi* is a simple puzzle that we’re going to be using as practice for 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 rightmost 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 leftmost 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.

These last two restrictions are what makes the puzzle difficult. If you’d like to learn more about the puzzle, you can read its wikipedia page here. Or if you’d like to try playing around with it yourself, this site provides an interactive version of the puzzle.

Your lab next week will be implementing a recursive solver for the Towers of Hanoi problem, so make sure you understand the concepts covered in this exercise!

Game Board Representation

If we were to write a program to simulate this puzzle, we would need some way to represent the location of each disk on the pegs. We can use a 2D array to represent the game board— since we know there can be at most $N$ disks (the total number of disks) on a peg and that there are exactly three pegs, any possible board state can be represented using a $3 \times N$ array of integers. If we use that representation, the value of $\text{board}[2][0]$ would be the size of the disk on peg 2 at the bottom (index zero).

We can create a 2D array like the one described above as follows:

```c
int **create_board() {
    int* board = new int[3];
    for (int i = 0; i < 3; i++) {
        board[i] = new int[NUM_DISKS];
    }
    return board;
}
```

Previously, we’ve used a variable of type `int*` to represent arrays. However, the array this function creates and returns seems to have the type `int**`— that is, a pointer to a pointer to an integer!

For this part of the exercise, first answer the following question: Why is the return type for this function `int**`? Could you create the function so that it returns a variable of type `int*` instead? Why or why not?

Recursive Puzzle Solution

One recursive solution to the Towers of Hanoi puzzle looks like this:

```c
//Recursive towers of hanoi solver
void towers(int disks, int **board, int source, int dest, int spare) {
    //Base case: If we're not being asked to move any disks, just return
    if (disks == 0) return;
    //First recursive call
towers(disks-1, board, source, spare, dest);
    //Move the topmost disk from the source peg to its destination peg
    move(board, source, dest);
    //Second recursive call
towers(disks-1, board, spare, dest, source);
}
```

If you call `towers(3, board, 2, 0, 1);` after setting up `board` to represent the puzzle’s initial state with three disks (where 2 is the index of the leftmost starting column and 0 is the rightmost goal column), this function will solve the puzzle one move at a time for you! It might be tough to see why the function works, so for this part of the exercise you should trace through a complete execution of the `towers(3, board, 2, 0, 1);` function solving the puzzle on a piece of paper, showing every move it makes. This can be done by manually walking through the code step-by-step, keeping track of each variable’s value, and moving a peg appropriately whenever the `move` function is called.

After you’ve finished explain how the recursive function solves the puzzle. What is the purpose of the first and second recursive calls? Why are they necessary?

For take-home exercises completed in peer-led groups, each student must participate in the class discussion and write answers to each of the questions on his/her own paper to show for credit.

For take-home exercises completed on your own, turn in your work electronically using the TEACH website.