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. You can earn a maximum of 3 points for lab work completed outside of lab time, but you must finish the lab before the next lab. For extenuating circumstances, contact your lab TAs and Jennifer Parham-Mocello.

Reminder: All of our labs involve paired programming. You do not have to keep the same partner for each lab, but you MUST work with someone in each lab!!! First, find a partner for this lab. It can be the same partner from the previous lab or a different partner.

(5 pts) Implement Towers of Hanoi

Now that you have drawn the picture and understand the game from exercise #8, design a solution to the program, including the functions and variables you’ll need.

First, you can implement your design using a static 2-D array with 3 columns for the 3 posts, 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 and a 2 x 3 board, i.e. call to towers(2, b, 1, 2, 3);

\[
\begin{array}{ccc}
1 & 0 & 0 \\
2 & 0 & 0 \\
\hline \\
0 & 0 & 0 \\
2 & 0 & 1 \\
\hline \\
0 & 0 & 0 \\
0 & 2 & 1 \\
\hline \\
0 & 1 & 0 \\
0 & 2 & 0 \\
\hline
\end{array}
\]

Here is an outline of the recursive towers function:
void towers(int disks, int b[][COLS], int from_col, int to_col, int spare)

    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)

(3 pts) Use Dynamic Array

Since this algorithm works for any n, then let's make our array dynamic. You will always have 3 columns, but you will now prompt the user for the number of rows. You will dynamically create a 2-d array for the Towers function to use. Write a function called create_array(). This function will take the number of rows as input and return a 2-d array on the heap: int ** create_array(int rows);

This function needs to create the rows and columns on the heap, but in order to do this, you have to create the row pointers and then each row with a specific number of columns. Here is how you would create the dynamic 2-d array for towers:

    int **b;   //a 2-d array is type **
    b=new int[rows];  //create row pointers
    //for each row pointer create the row with 3 cols
    for(int i=0; i<rows; i++)
        b[i]=new int[3];  //create the row with 3 cols

Now, change towers() to work with a dynamic 2-d array, which is int **b parameter instead of int b[][COLS]. Get towers working with n disks using a n X 3 board.

(2 pts) Command-line Arguments

Change the program to get the number of rows using a command line argument, instead of prompting the user for this information. Command line arguments are supplied at the time you run your program:

    lab8.exe 3

By default, every program has one command-line argument, which is the name of the program you ran. You can access the command-line arguments by adding two parameters, one for the number of arguments and the other for the arguments as an array of c-style strings, to the main() function. For example:

    int main(int argc, char *argv[])

By default, the argc is one and argv[0] is a c-style string corresponding to the name of the program because the name of the program is always the first argument.