Roleen Ferrer 1731261

Professor Maxwell Dunne

CSE 13S - Computer Systems and C Programming

Assignment 3: Tower of Hanoi

For this lab, I was able to correctly implement algorithms in solving the Tower of Hanoi problem.

Recursion (Recursive)

While thinking about how to implement the recursive algorithm, I had to think about how to make the function run recursively. I thought of many ways, and eventually I found a pattern that when called correctly, will solve the problem with any n disks. I first thought about what my exit condition would be, which would be when the top of the disk (or 1) is at the top of peg B when calling this recursive function. By having a parameter that takes the number of total disks, the function will run continuously subtracting by n-1 each time I were to switch the order of the pegs in the recursive function. Calling this continuously will eventually solve the problem, and it only took some time to figure out where the correct printf statements would be located in the function. I also implemented another case where the total number of disks would = 0, which prints that there were no disks.

Iterative (Stack)

While thinking about how to implement the iterative stack function, I first had to implement the stack functions in stack.c which I would use in my tower.c source code. I had a lot of help from using the lecture notes in implementing these functions, and while the code supplemented was not entirely exact, it helped me understand the functionality between each function and the proper memory allocation needed in order to produce a stack. For the algorithm the stacks would run by, I thought of the pattern of legal moves that a disk can move from column to column. I noticed that depending on the total number of disks, if they were even or odd, they would work somewhat oppositely. For example, the legal moves for an odd number of total disks and if the current disk is odd are from A-B, C-B, and C-A (see DESIGN.pdf for more information on this algorithm). When the problem is solved, I would then delete each stack using a function from stack.c in order to free up the memory and prevent a memory leak.

Conclusion

This lab taught me a lot on the implementation of certain functions in order to complete a task. Both of the iterative and recursive functions would give out the same output, regardless of input. Yet, as for time complexity, I believe that the stack implementation would be faster than the recursive function. For disks with inputs that are over 10, the recursive function would have to call itself numerous times, which is very inefficient in comparison to a stack function, which keeps track of where each disk is and sufficiently solves the problem with an iterative loop. Recursive functions are basically a "divide and conquer" type algorithm, as seen in lecture, which inherently is very inefficient at times. Using a stack can also provide a clearer understanding in code, as the stack is able to be accessed through various pointer calls (e.g stack->items). This makes it easier to debug code that was parsed through the stack, in comparison to a recursive function that may require some complex debugging if done incorrectly. Stacks can be compared to arrays, which can hold data that is more organized than solving a problem recursively.