COP 3502 Suggested Program Edits: Recursion (Week 2 Programs)

1) Look up the Lucas Numbers and write a recursive function that takes in a single non-negative integer, n, and returns the n^{th} Lucas Number.

2) Write the Tip Chart function shown in class recursively, but with a different breakdown. Here are a couple ideas:

> a) Recursively print the whole chart but the last row, THEN print the last row.
> b) If there's one row left, print it. Otherwise, print the first half of the chart, followed by the second half of the chart.

3) A binomial combination is defined as follows:

> C(n, 0) = C(n, n) = 1, for all non-negative integers n
> C(n, k) = C(n-1, k-1) + C(n-1, k), for all cases where 1 <= k <= n-1.

Write a recursive function that takes in n and k as parameters and returns C(n, k), using the recursive characterization shown above. (Note: You can check if you did this correctly because we also know that C(n, k) = n!/k!/(n-k)!, so you can use your factorial code to double check this code!)

4) Write a function to find the maximum value in an array recursively, but instead of using the strategy shown in class, use strategy (b) from problem #2 above.

5) Consider the problem of a frog jumping out of a well. Initially, the frog is ***n*** feet below the top of the well. When the frog jumps up, it elevates ***u*** feet. If a jump gets the frog to the top of the well or past it, the frog escapes the well. If not, unfortunately, the frog slips down by ***d*** feet before clinging to the side of the well. (Note that ***d*** < ***u***.) Write a ***recursive*** function that takes in positive integers, ***n***, ***u***, and ***d***, and returns the number of times the frog must jump to get out of the well.

For example, if n = 10, u = 5 and d = 3, the function should return 4. On the first jump, the frog goes from 10 feet below the top to 8 feet below (5-3 is the progress). On the second jump, the frog goes from 8 feet below the top to 6 feet below the top. On the third jump, the frog goes from 6 feet below the top to 4 feet below the top. On the last jump, since 5 feet is enough to clear the top of the well, the frog does not slip down and gets out. In this case, had n = 11, the frog would have also gotten out in 4 jumps.

(Note: Although one can do some math to arrive at an O(1) solution without recursion, please use recursion to simulate the jumping process described as this is what is being tested - the ability to take a process and express it in code, recursively. Also, though this is a toy problem, it's surprisingly similar to the real life process of paying off a loan, though in the latter process, the amount you "slip down" slowly decreases, month after month.)

```
int numJumps(int n, int u, int d) {
// Fill in code here.
}
```