<u>COP 3502 Quiz #2 Version A (Recursion, Linked Lists, Stacks, Queues) Solutions</u>

1) (5 pts) Write a **<u>recursive function</u>** that takes in an integer array, values, and an integer k, and returns the sum of the even integers in the array in between indexes 0 and k-1, inclusive. For example, if the input to the function was the array values = [3, 5, 2, 9, 4, 8, 7] and the integer k = 5, then the function should return 6, since out of 3, 5, 2, 9 and 4, only 2 and 4 are even. The function prototype is provided for you below. **(Note: Please do NOT put any printfs, scanfs or loops in your response. The presence of any of these will automatically earn 0 points for this question.)**

```
int sumEvenValues(int* values, int k) {
    if (k == 0) return 0;
    int res = 0;
    if (values[k-1]%2 == 0) res += values[k-1];
    return res + sumEvenValues(values, k-1);
}
```

Grading: base case - 1 pt, rec call with k-1 - 2 pts, 1 pt for odd case not add, 1 pt for even case adding

2) (8 pts) Convert the following infix expression to postfix using a stack. Show the contents of the stack at the indicated points (A, B, and C) in the infix expression and provide the corresponding postfix expression as well.

4 * (2 + 12 / (1 A + 10 / 2) B - 1) + 10 - (2 * (1 C + 3))

A	B	C
(		
/	/	(
+	+	*
(	(	(
*	*	-

Resulting postfix expression:

4	2	12	1	10	2	/	+	/	+	1	-	*	10	+	2	1	3	+	*	-

Grading: 1 pt for each stack, 5 pts for the expression, take off 1 pt for each error in the expression (a single error is usually an item out of place, so to fix it, you insert an item in the different place. The fewest # of insertions needed to fix the expression is the number of points to take off, capped at 5)

3) (10 pts) Jayson considers odometer readings on his car which have the same digit appearing consecutively to be bad luck. He has figured out how to adjust his car's odometer (this is illegal, by the way) so that it automatically skips over any settings which have the same digit appearing in consecutive slots. (For example, if his odometer had 5 digits, it would start at the setting 01010, the smallest integer that can be represented without placing the same digit twice in a row. The next setting would be 01012.) Complete the program below so that it prints out, in order, all the odometer settings Jayson's car will hit. (You may assume that we use the 10 digits 0 through 9 and that his odometer has n spots, where n is a parameter passed into the recursive function. The initial call to the recursive function is provided in the completed wrapper function. Please call the print function to print a single odometer setting. k represents the number of slots of the odometer that are fixed.)

```
#define NUMSLOTS 5

void printJayson() {
    int odometer[NUMSLOTS];
    printJaysonRec(odometer, 0, NUMSLOTS);
}

void printJaysonRec(int odometer[], int k, int n) {


    if (k == n) {

        print(odometer, n) ;      // Grading: 2 pts

        return ;                  // Grading: 1 pt
    }

    for (int i=0; i<10; i++) {

        if ( k>0 && odometer[k-1] == i ) continue; // 1 pt, 2 pts

        odometer[k] = i ;                          // 2 pts

        printJaysonRec(odometer, k+1, n) ;         // 2 pts
    }
}

void print(int array[], int n) {
    for (int i=0; i<n; i++)
        printf("%d", array[i]);
    printf("\n");
}
```

4) (12 pts) Write a function that takes in a pointer to a linked list, finds the last node of the linked list, places that node in the front of the linked list and returns a pointer to the new front of the list. (If the list has fewer than two elements a pointer to the list is returned and no changes are made.) Use the struct definition given below and fill in the given function prototype.

```
typedef struct node {
    int data;
    struct node* next;
} node;

node* backToFront(node* listPtr) {

    if (listPtr == NULL || listPtr->next == NULL)
        return listPtr;

    node* back = listPtr;
    while (back->next->next != NULL)
        back = back->next;

    node* newfront = back->next;
    back->next = NULL;
    newfront->next = listPtr;
    return newfront;

}
```

Grading: Base cases - 3 pts total
Iterating to second to last node - 3 pts (2 pts if go to last node)
Patching last to first - 2 pts
Making new last node (NULL) - 2 pts
Returning the new front of the list - 2 pts