COP 3502 Quiz #2 Version B (Recursion, Linked Lists, Stacks, Queues) Solutions

1) (5 pts) Write a **recursive function** that takes in an string word, and an integer k, and returns the number of occurrences of the character 'a' in the first k letters of the string. (You may assume that k is less than or equal to the length of the string word.) For example, if the input to the function was the string word = "abracadabra" and the integer k = 6, then the function should return 3, since "abraca" contains 3 occurrences of 'a'. 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 countAs(char* word, int k) {
    if (k == 0) return 0;
    int res = 0;
    if (word[k-1] == 'a') res++;
    return res + countAs(word, k-1);
}
```

Grading: base case - 1 pt, rec call with k-1 - 2 pts, 1 pt for not 'a' don't add, 1 pt for 'a' 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.

3 * (4 + 16 / (2 + 18 / 3) - 1) + 8 - (2 * (3 + 2))

(A, B, C mark the positions in the expression above)

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

Resulting postfix expression:

3	4	16	2	18	3	/	+	/	+	1	-	*	8	+	2	3	2	+	*	-

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) What are the first 10 lines of output of the following program? (Note: do not try to trace the program line by line. Try to understand what the code is doing systemically.)

```
#include <stdio.h>
#define SIZE 4
void go(int* perm, int* used, int k, int n, int ok);
void print(int array[], int n);
int main(void) {
    int perm[SIZE], used[SIZE];
    for (int i=0; i<SIZE; i++) used[i] = 0;
    go(perm, used, 0, SIZE, 0);
    return 0;
}

void go(int* perm, int* used, int k, int n, int ok) {
    if (k == n) {
        print(perm, n);
        return;
    }
    int canDo = 0;
    for (int i=k; i<n; i++)
        if (!used[i])
            canDo = 1;
    if (!ok && !canDo) return;
    for (int i=0; i<n; i++) {
        if (used[i]) continue;
        used[i] = 1;
        perm[k] = i;
        go(perm, used, k+1, n, ok || k == i);
        used[i] = 0;
    }
}

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

0, 1, 2, 3,
0, 1, 3, 2
0, 2, 1, 3
0, 2, 3, 1
0, 3, 1, 2
0, 3, 2, 1
1, 0, 2, 3
1, 2, 0, 3
1, 3, 2, 0
2, 0, 1, 3

Grading: 1 pt per line, in order. If they list all perms in order (not skipping 1,0,3,2 etc.) 6 out of 10.

4) (7 pts) Consider storing an integer in a linked list by storing one digit per node, in reverse order. For example, the linked list → 3 → 2 → 8 → 5 represents the integer 5823. Write a **recursive function** that takes in a linked list stored in this fashion and returns the value of the integer being stored. (You may assume no overflow errors will occur.)

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

int getValue(node* listPtr) {
    if (listPtr == NULL) return 0;
    return listPtr->data + 10*getValue(listPtr->next);
}
```

Grading: base case - 2 pts, 1 pt - isolating units digit, 1 pt mult 10, 2 pts rec call, 1 pt return

5) (5 pts) Using the same storage system for an integer as described in question 4, write a **recursive function** which prints out the integer represented by the linked list pointed to by listPtr. For example, if the list pointed to by listPtr was → 3 → 2 → 8 → 5, then your function should print 5823.

```
void print(node* listPtr) {

    if (listPtr != NULL) {
        print(listPtr->next);
        printf("%d", listPtr->data);
    }

}
```

Grading: 1 pt for not getting a NULL ptr error on a null list,
1 pt for printing data
2 pts for recursive call
1 pt for ORDER of rec call and printf