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

1) (5 pts) Write a recursive function that returns the **product** of the digits of its input value, ***n***. You may assume that ***n*** > 0. For example, prodOfDigits(37) should return 21, prodOfDigits(8) should return 8 and prodOfDigits(309823) should return 0.

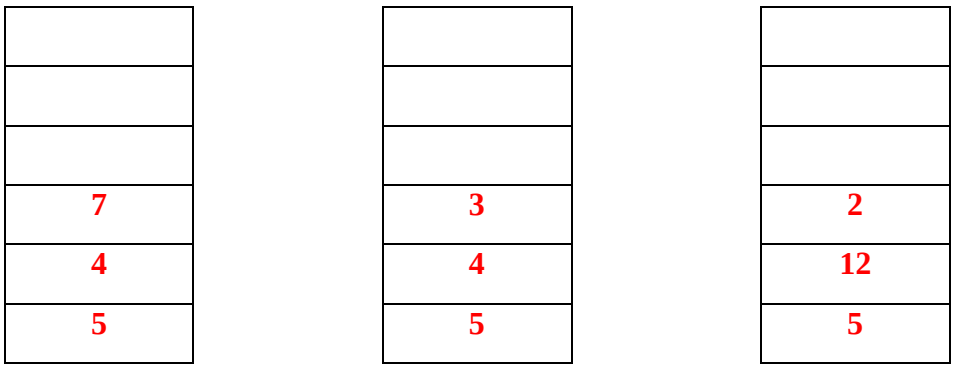
```
int prodOfDigits(int n) {

    // Grading: 2 pts - also give credit for if (n==0) return 1.
    if (n<10) return n;

    // 1 pt n%10, 1 pt rec call, 1 pt n/10
    return (n%10)*prodOfDigits(n/10);
}
```

2) (5 pts) Evaluate the following postfix expression, showing the state of the stack at each of the indicated points.

```
                A                B                C
2  3  +  4  7  2  2  *  -  *  6  3  /  /  +
```



A B C

Value of the Expression: **11**

Grading: 1 pt for each stack (all or nothing), 2 pts for the final answer (all or nothing)

3) (8 pts) Consider using a linked list of digits to store an integer. Specifically, the integer 2471 would be stored as the following linked list: 1 → 7 → 4 → 2. (Thus the first node stores the units digit, the second node the ten's digit, the third node the hundred's digit and so forth.) ***Complete the recursive*** function below that takes in two pointers to integers stored in this manner, num1 and num2, and returns a negative integer if the integer represented by num1 is smaller than the one represented by num2, 0 if the two integers are equal, and a positive integer if the integer represented by num1 is larger than the one represented by num2.

```
typedef struct numNode {
    int digit;
    struct node* next;
} numNode;

int compare(numNode* num1, numNode* num2) {
    if (num1 == NULL && num2 == NULL) return 0;
    if (num1 == NULL) return -1;
    if (num2 == NULL) return 1;

    // Grading: 3 pts
    int tmp = compare(num1->next, num2->next);
    // Grading: 2 pts
    if (tmp != 0) return tmp;
    // Grading: 3 pts
    return num1->digit - num2->digit;
}
```

4) (5 pts) What problem does the function below solve? Namely, what is the return value of the function in relation to the input parameters? Your answer should be a regular sentence or two in English and NOT a literal translation of the code. The latter will receive **no credit!**

```
// Pre-condition: array is size n and sorted with unique values.
//                target > 0.
int mysteryB(int* array, int n, int target) {

    int res = 0, low = 0, high = 0;
    while (high < n) {
        if      (array[high] - array[low] < target) high++;
        else if (array[high] - array[low] > target) low++;
        else {
            res++;
            low++;
            high++;
        }
    }
    return res;
}
```

This function determines the number of pairs of integers in the array that differ by exactly target. Note: An individual value may be part of two separate pairs. For example, if the array stores 3, 9 and 15 and the target value is 6, then both (3, 9) and (9, 15) are counted.

Grading: 2 pts for mentioning difference of values in array, 2 pts for mentioning a running tally of some sort, 1 pt for putting it all together

5) (5 pts) The permutation algorithm shown in class generates the permutations of {0, 1, 2, …, , n-1} in lexicographical order. Which permutation does the algorithm generate after the permutation shown below?

2, 9, 6, 3, 8, 7, 5, 4, 1, 0

2, 9, 6, 4, 0, 1, 3, 5, 7, 8

C

6) (7 pts) In the implementation of a queue using an array shown in class, the following four components formed the structure storing the queue:

```
elements - an array storing the items in the queue.
front - index of the array that is the front of the queue
numElements - number of elements currently in the queue.
queueSize - currently allocated space for array elements.
```

Assume that the queueSize is initially set to 10 and that the following code segment is run:

```
struct queue* MyQueuePtr = malloc(sizeof(struct queue));
init(MyQueuePtr);

for (int i=0; i<8; i++) {
    enqueue(MyQueuePtr, 2*i+1);
    enqueue(MyQueuePtr, 6*i+2);
    int tmp = dequeue(MyQueuePtr);
}
```

Show the value of each of the four components of the struct pointed to by MyQueuePtr right after this code segment is executed. **Note: for the array, ONLY fill in the slots that have meaningful data. Leave blank any spots that aren't storing something in the queue.**

front **8** numElements **8** queueSize **10**

index	0	1	2	3	4	5	6	7	8	9
elements	**11**	**32**	**13**	**38**	**15**	**44**			**9**	**26**

Grading: 1 pt for each of front, numElements and queueSize
1/2 pt for each of the other 8 entries, round down
So only give 7/7 if everything is correct. 2 of the entries in the array can be wrong to get 6/7