COP 3502 Quiz #2 Version C (Recursion, Linked Lists, Stacks, Queues) Solution

1) (5 pts) Write a **recursive function** that takes in an integer array, values, an integer k, and an integer max, and returns 1 if the first k values in the array are sorted and less than or equal to max, or 0 otherwise. For example, if the input to the function was the array values = [2, 4, 5, 5, 8, 6] and the integers max = 8 and k = 5, then the function should return 1 since [2, 4, 5, 5, 8] is sorted and all values are less than or equal to 8. (But, if we passed in the same array and max = 7 and k = 5, the function should return 0.) 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 isSorted(int* values, int k, int max) {

    if (k == 0) return 1;
    if (values[k-1] > max) return 0;
    return isSorted(values, k-1, values[k-1]);

}
```

Grading: base case k == 0: 1 pt, base case last value above max: 1 pt, recursive call 3 pts (1 pt for return and func call, 1 pt for k-1 and 1 pt for values[k-1])

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.

5 * (2 + 20 / (1 [A] + 24 / 6) [B] - 1) + 6 - (3 * (2 [C] + 5))

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

Resulting postfix expression:

5	2	20	1	24	6	/	+	/	+	1	-	*	6	+	3	2	5	+	*	-

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) Consider implementing a stack with a dynamically sized array. If an attempt is made to push an item onto the stack that is currently full, then the array storing the stack values can be reallocated to be twice its current size. To make coding this a bit easier, assume that the realloc always succeeds. (No need to check if it returns NULL.) Complete the push function for the stack. Some supporting code is given which you will need to carefully read to figure out details about the implementation. Your implementation of push must be consistent with the rest of the code provided.

```
typedef struct stack {
    int* stackItems;
    int capacity;
    int top;
} stack;

void init(stack* sPtr, int initSize) {
    sPtr->stackItems = calloc(initSize, sizeof(int));
    sPtr->capacity = initSize;
    sPtr->top = 0;
}

// Pre-condition: stack pointed to by sPtr is NOT empty.
int pop(stack* sPtr) {
    sPtr->top--;
    return sPtr->stackItems[sPtr->top];
}

void push(stack* sPtr, int value) {

    if (sPtr->top == sPtr->capacity)
        sPtr->stackItems =
        realloc(sPtr->stackItems, 2*sPtr->capacity*sizeof(int));

    sPtr->stackItems[sPtr->top] = value;
    sPtr->top++;

}
```

Grading: 2 pts check if full,
5 pts realloc (1 pt LHS, 1 pt realloc, 1 pt pointer, 1 pt sizeof, 1 pt - 2*sPtr->cap)
2 pts add value (1 pt for sPtr-> total, 1 pt for index top, 1 pt for value)
1 pt increment top

4) (12 pts) Consider storing a string in a linked list by storing one character per node. For example, the linked list → d → o → o → r represents the string "door". Write a **iterative function** which takes in a pointer to a linked list storing a string in this fashion and returns a dynamically allocated character array storing the contents of the string. Allocate precisely the appropriate number of characters for the array. Don't forget to null terminate the string before you return it! (Note: You will have to make two passes through the list: the first pass will just determine the length of the linked list. Once you know this length, you can allocate the string and copy in the appropriate contents into the string from the linked list.)

```
typedef struct charnode {
    char letter;
    struct charnode* next;
} charnode;

char* convert(charnode* word) {

    int len = 0;
    charnode* tmp = word;
    while (tmp != NULL) {
        tmp = tmp->next;
        len++;
    }

    char* res = malloc(sizeof(char)*(len+1));

    for (int i=0; i<len; i++) {
        res[i] = word->letter;
        word = word->next;
    }

    res[len] = '\0';
    return res;

}
```

Grading: 4 pts obtaining length of the linked list.
2 pts malloc char array
4 pts copy letters in
1 pt null terminate
1 pt return