COP 3502 Recitation Sheet: Linked List Solutions

Directions: Each of these questions is from either a past Foundation Exam or one of my past exams. They were created to be written on paper. However, each of these can be coded on a computer and tested. I strongly recommend first coding on paper, but then transferring to the computer and testing the function until you are convinced it works.

Each of the following questions asks you to write a recursive function.

1) An alternate method of storing a string is to store each letter of the string in a single node of a linked list, with the first node of the list storing the first letter of the string. Using this method of storage, no null character is needed since the next field of the node storing the last letter of the string would simply be a null pointer. Write a function that takes in a pointer to a linked list storing a string and returns a pointer to a traditional C string storing the same contents. Make sure to dynamically allocate your string in the function and null terminate it before returning the pointer to the string. Assume that a function, length, exists already that you can call in your solution, that takes in a pointer to a node and returns the length of the list it points to. The prototype for this function is provided below after the struct definition.

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

int length(node* head);

char* toCString(node * head) {

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

    int i=0;

    while (head != NULL) {
        res[i] = head->letter;
        head = head->next;
        i++;
    }

    res[len] = '\0';

    return res;
}
```

2) Suppose we have a linked list implemented with the structure below. Write a function that will take in a pointer to the head of a list and inserts a node storing -1 ***after*** each even value in the list. If the list is empty or there are no even values in the list, no modifications should be made to the list. (For example, if the initial list had 2, 6, 7, 1, 3, and 8, the resulting list would have 2, -1, 6, -1, 7, 1, 8, -1.)

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

void markEven(node *head) {

    node* tmp = head;

    while (tmp != NULL) {

        while (tmp != NULL && tmp->data%2 != 0)
            tmp = tmp->next;

        if (tmp != NULL) {
            node* newnode = malloc(sizeof(node));
            newnode->data = -1;
            newnode->next = tmp->next;
            tmp->next = newnode;
            tmp = newnode;
        }
    }
}
```

3) Suppose we have a linked list implemented with the structure below. The function below takes in a pointer, **head**, to a linked list which is guaranteed to store data in strictly ascending order. If the list doesn't contain the value 3, the function should create a struct node storing 3 in its data component, insert the node so that the listed pointed to by head stores its data, including 3, in strictly ascending order, and returns a pointer to the front of the resulting list. If a node already exists storing 3 in the list pointed to by head, then return head and make no changes to the list.

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

node* addValue3(node* head) {

    if ( head == NULL || head->data > 3 ) {
        node* tmp = malloc(sizeof(node));
        tmp->data = 3;
        tmp->next = head;
        return tmp;
    }

    if ( head->data == 3 )
        return head;

    node* iter = head;
    while (iter->next != NULL && iter->next->data < 3 )

        iter = iter->next;

    if ( iter->next != NULL && iter->next->data == 3 )
        return head;

    node* tmp = malloc(sizeof(node));
    tmp->data = 3;
    tmp->next = iter->next;
    iter->next = tmp ;
    return head;
}
```

4) Given the linked list structure named node, defined in lines 1 through 4, and the function named eFunction defined in lines 6 through 14, answer the questions below.

```
 1 typedef struct node {
 2         int data;
 3         struct node * next;
 4 } node;
 5
 6 node* eFunction(node* aNode){
 7    if(aNode == NULL) return NULL;
 8    if(aNode->next == NULL) return aNode;
 9
10    node* rest = eFunction(aNode->next);
11    aNode->next->next = aNode;
12    aNode->next = NULL;
13    return rest;
14 }
```

(a) (1 pt) Is this function recursive? (Circle the correct answer below.)

[✓ **YES**]   NO

(b) (2 pts) What does the function eFunction do, in general to the list pointed to by its formal parameter, aNode?

This function reverses the list originally pointed to by aNode and returns a pointer to the new front of the list

(c) (2 pts) What important task does line 12 perform?

The last node in a linked list must have its next pointer point to NULL. That is how most linked list functions detect the end of the list. Line 12 does this since aNode ends up point to the last node in the list. After the reversal is complete, it's necessary to make sure that the next pointerof the last node in the resulting list is pointing to NULL because before line 12 it's not. (It's pointing to the second node in the original list, which is the last node in the list pointed to by rest.)

5) The structure of each node of a singly linked list is shown below.

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

Write a function insertAfterN, that takes the head of a linked list, and two integers M and N (M ≠ N) and inserts M after all the nodes containing N.

For example, if M = 200 and N = 6, the linked list 3, 6, 4, 6, 6, 5 will be changed to 3, 6, 200, 4, 6, 200, 6, 200, 5.

```
void insertAfterN(node* head, int M, int N) {

    if (head == NULL) return;

    if (head->data == N) {
        node* tmp = malloc(sizeof(node));
        tmp->data = M;
        tmp->next = head->next;
        head->next = tmp;
        head = tmp;
    }

    insertAfterN(head->next, M, N);

}
```