**Programming Project 2 (40 points)**
**Due: See Canvas for due date**
Vsp.2022

## Assignment Description

In this assignment you will implement and manipulate a linked list using the C programming language. The operations on a linked list you will implement will offer you the opportunity to develop your ability with complex pointer algorithms.

## Implementation Requirements

Your assignment must meet at least the following requirements:

- Your program must be written in C.
- All functions/code must use dynamic memory when using and manipulating linked list.
- Before exiting, your program must release all memory allocated. (In the main() offer an option to terminate the program)
- Create a "linkedlist.h" file for all struct and function prototypes. Implement all the functions in a "linkedlist.c" file that includes the "linkedlist.h" file.
- Create a "main.c" file that includes the "linkedlist.h" file and write code to use ALL your functions so YOU can test the correct functionality of each function. (You can include a numbered-based menu; one option for each function)
- Use the following struct:

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

## Operations to implement

1. **void Push**(**Node** headRef, int newData**).- Write a Push() function that given an *int* and a reference to the head pointer (i.e. a struct node** pointer to the head pointer), add a new node at the head of the list with the standard 3-step-link-in: (i) create the new node, (ii) set its *next* to point to the current head, and finally (iii) change the head to point to the new node.

2. **int Pop(Node** headRef)**.- Write a Pop() function that is the inverse of Push(). Pop() takes a non-empty list, deletes the head node, and returns the head node's data. Pop() should fail if there is not a node to pop. (*If all you ever used were Push() and Pop(), then our linked list would really look like a stack*)

3. **int Count(Node* head, int searchFor)**.- Write a Count() function that counts the number of times a given int occurs in a list.

4. **int GetNth(Node** head, int index)**.- Write a GetNth() function that takes a linked list and an integer index and returns the data value stored in the node at that index position. GetNth() uses the C numbering convention that the first node is index 0, the second is index 1, ... and so on. So for the list {42, 13, 777} GetNth() with index 1 should return 13. The index should be in the range [0 .. length-1]. If it is not, GetNth() should fail (or you could implement some other error case strategy).

5. **void DeleteList**(**Node** head**).- Write a function DeleteList() that takes a list, deallocates all of its memory and sets its head pointer to NULL (the empty list).

6. **void InsertNth**(**Node** head, int index, int data**).- Write a function InsertNth() which can insert a new node at any index within a list. The caller may specify any index in the range [0..length], and the new node should be inserted so as to be at that index.

7. **void MoveNode**(**Node** destRef, Node** sourceRef**).- MoveNode() takes two lists, removes the front node from the source list and pushes it onto the front of the destinationlist.

8. **void Recursive Reverse**(**Node** headRef)**.- Write an recursive Reverse() function that reverses a list by rearranging all the *.next* pointers and the head pointer. Reverse() should only need to make a single pass over the list (Doing it with multiple passes is easier but very slow).

## Submission

Your project must be submitted using the Assignment Submission Link in Canvas. You will see this link when you go to Assignments page. Submit a zip file of your project by the specified due date. If you use CLion, please zip the folder that includes the CLion project.

**For this project, Canvas will accept .zip files ONLY**.

### Grading

Programs that do not compile will receive a grade of 0. A grading breakdown for programs that do compile appears below:

| | |
|---|---|
| Correct Push()function | 3 |
| Correct Pop() function | 3 |
| Correct Count() function | 3 |
| Correct GetNth() function | 3 |
| Correct DeleteList() function | 3 |
| Correct InsertNth() | 5 |
| Correct MoveNode() | 5 |
| Correct recursive Reverse() function | 15 |
| **Total** | 40 |