Lab 2: Linked Lists

ELEC 278: Fundamentals of Information Structures

Learning Goals

- Gain a solid understanding of the basic structure of linked lists, including concepts like nodes, pointers, head, tail, and cycles.
- Develop the ability to implement core linked list operations, such as insertion, deletion, and traversal.

Instructions

Download the zipped folder `lab2.zip` from OnQ and unzip it. Next, open the lab2 folder that you extracted in your code editor. You will complete the following exercises by completing the functions in the `main.c` file.

To receive full marks, your code must be able to correctly handle any list of any length and correctly handle corner cases. Comment your code and be ready to discuss how your code handles corner cases with your TA. Generate a list a cases that you will test your code against with your TA.

Grading (fraction of exercise marks)

0	No code.
1/2	Functioning code, but fails corner cases, if any, or not readable/not well-explained.
1	Functioning code that handles corner cases

Exercise 1. (0.5 mark)

You are given the following C-struct for a linked list node.

```
struct Node {
    int val;
    struct Node* next;
};
```

Write a function `void push(struct Node** head, int new_val)` to create a linked list. `push` adds a new node to the front of the list.

Exercise 2. (0.5 mark) Write a function `void printList(struct Node *node)` to print the entire list.

Exercise 3. (1 mark) Write a function `struct Node* deleteDuplicates(struct Node* head)`, which given the head of a sorted single linked list, deletes all **consecutive** duplicates such that each element appears only once. Return the head of the modified list.

For example, given a linked list $1 \to 1 \to 2$, return a linked list $1 \to 2$

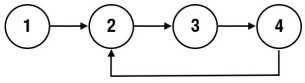
Exercise 4. (1 mark) Write a function `struct Node* reverseList(struct Node* head)`, which given the head of a singly linked list, reverses the list, and return the new head of the list.

For example, given a linked list $1 \to 2 \to 3$, return a linked list $3 \to 2 \to 1$

Exercise 5. (1 mark) Write a function `bool hasCycle(struct Node* head)`, which given the head of a linked list, returns true if there is a cycle in the linked list; otherwise, returns false.

There is a cycle in a linked list if at least one node in the list can be visited again by following the next pointer.

For example, the following linked list is cyclic and should return true.



A linked list $1 \to 2 \to 3 \to 2 \to 1$ returns false.

Hint: can you use more than one pointer to traverse the linked list?

Exercise 6. (1 mark) Write a function `bool isPalindrome(struct Node* head)`, which given the head of a singly linked list, returns true if it is a palindrome or false otherwise.
A palindrome is a sequence that reads the same forward and backward.

For example, return `true` for a linked list $1 \to 2 \to 3 \to 2 \to 1$, and `false` for a linked list $1 \to 2 \to 3$

Hint: can you re-use the reverseList function?