Assignment 3

DATA 311
Due: **Nov. 2, 2019**

Ground rules:

- Code should be compatible with Python 3,
- Submission *should* be **ONE** file in {.py or .ipynb}format.
- Any non-trivial functions used in your code should be written by you. *Non-trivial*, in this context, is defined as anything that performs a significant operation in your code. You can freely use math operators, loop and conditional statements without re-defining them.
- You are allowed to refer to external resources or collaborate on the assignment. However, any work submitted must be compliant to the University Academic Integrity guidelines, which can be found here, and any resources used *must* be listed.
- Solutions to this assignment will be discussed on **Nov. 04, 2019** during lab.
- The assignment is worth **100 points**, which will be scaled to **10%** of the final grade.

In this assignment, you will create a calculator for postfix expressions. The ideas required for this question are covered in lectures talking discussing slide sets 8 and 9, and during lab.

Again, to emphasize, the code for all parts of this assignment should be contained in a *single* `.py` or `.ipynb` file.

Note that the data structures you create should be able to support float and character data types.

Creating a doubly linked list (40 points)

Write a program to initialize a doubly linked list (in this question, the instance of a DLL will be referred to as `dll`), and perform the following functions on them:

1. `createDLL()` *(10 points)*: Should create an instance of a doubly linked list, `dll`. Function does not need to return anything.
2. `isEmpty(dll)` *(10 points)*: Takes as input an instance of a doubly linked list, `dll`, and checks if it is empty. Should return 1 if `dll` is not empty, and -1 if `dll` is empty.
3. `addFront(dll, val)` *(10 points)*: Takes as input an instance of a doubly linked list, `dll`, and a value, `val`, and adds it to the head of the list. Function does not need to return anything.
4. `removeFront(dll)` *(10 points)*: Takes as input an instance of a doubly linked list, `dll`, and removes the head of the list. Function does not need to return anything.

How to test your code: Make sure you print out the DLL whenever you perform `addFront` or `removeFront` functions.

Creating a stack from a doubly linked list (20 points)

Use the above doubly linked list code to create a stack. For the purposes of this problem, the instance of a stack will be referred to as `stack`. Your stack implementation should support the following functions:

1. `createStack()` *(4 points)*: Use the `createDLL()` function from the previous part to initialize an empty stack.
2. `push(stack, val)` *(8 points)*: Use the `addFront()` function from the previous part to implement the stack push function.
3. `pop(stack, val)` *(8 points)*: Use the `removeFront()` function from the previous part to implement the stack pop function.

How to test your code: Make sure to print out the stack after `push` and `pop` functions.

Evaluating postfix expressions (40 points)

Use the stack implemented in the previous part and use it to create a postfix expression calculator. The calculator should take as input an expression, `exp`, from the user, and check if `exp` is a valid postfix expression. If `exp` is a valid postfix expression, then compute it and print out the value that `exp` evaluates to. If `exp` is an invalid postfix expression, print out "User input is not a valid postfix expression".

You are not required to support brackets/parenthesis. However, if you do support brackets/parenthesis, then that should also be taken into account while checking the validity of the expression.

Bonus question - Performing O(log n) search in linked lists (40 points)

(**Note:** If this problem is attempted, the non-coding parts of the problem should be submitted as a `.pdf` file, where the file was produced as output of a word processing application and should consist only of typeset characters. Please name this file `bonus.pdf`. The coding part of this problem should be named `bonus.py` and all three files should be compressed and submitted as a `.zip` file.)

Assignment 3

DATA 311
Due: **Nov. 2, 2019**

In the last assignment, you implemented binary search, which is a $\mathcal{O}(\log n)$ algorithm. In this problem, we will see how to do it in a singly linked list.

Searching on a linked list (10 points)

What is the biggest drawback in a linked list (SLL or DLL) that makes $\mathcal{O}(logn)$ search impossible on it?

A candidate modification to linked lists (10 points)

Consider a doubly linked list. We know that every node in a DLL contains a pointer to the next and previous nodes respectively. Now suppose every node has a next pointer to every other node that appears on the list after it, and a previous pointer to every other node that appears on the list prior to it. Suppose this list has n such nodes.

1. What is the overall overhead in terms of pointer storage per node and on the list as a whole?
2. What is the average asymptotic cost for inserting a node at an arbitrary location in this list?
3. Describe how you might perform binary search on this modified list.

Implementing binary search on doubly linked lists (20 points)

Implement this modified version of doubly linked list. It should support the same operations the previous problem on the assignment supported, as well as binary search. Of course, it is beneficial to store pre-sorted values in this list.