Data Structures Homework 03

Important:

- Download the hw03.cpp file from Canvas -> Assignments -> Homework03.
- Write your code (**in C++**) in the designated positions in hw03.cpp
- The purpose of the homework is to practice using some of the concepts gone over in class. **The problems in the first two functions must be solved using stacks.** You may use std::stack. If you don't solve them with the stack, no points will be given.
- To submit, upload hw03 to Gradescope. Because of the way the autograder on Gradescope is configured, the file must be named hw03; otherwise, the test cases will not run and you will not get any points.
- Only some of the functions will be for code. It may be different for each student.
- The main portion of your points will come from your explanation. Please make sure to document each line of code and write a small statement explaining your thought process underneath the function in the comments.
- You may upload as many times as you want before the deadline. Each time the autograder will let you know how many of the test cases you have passed/failed. To prevent people from manipulating the autogader, the content of half of the test cases are hidden.
- Please **do not copy others' answers**. Autograder can detect similar code. Also, for your own benefit, do not try to find solutions online.

Function 1: postfixEval(arr)

Write a function named postfixEval such that given a postfix expression, evaluate it and return the result.

Assumptions:

- the operands are floats
- the possible operators are +, -, *, /
- the input expression is always a valid and none-empty postfix expression

Example: When input is ["2", "4", "7", "∗", "−"], which is $2 - 4 * 7$ using infix notation, your function should -26.

Function 2: validParentheses(arr)

Given a (possibly empty) array containing only alphabet letters and parentheses "(", ")", "{", "}", "[", "]", determine if the string is valid.

A string is valid if

- Open brackets must be closed by the same type of brackets.

- Open brackets must be closed in the correct order.

Example: Given input “(ab){[]}”, your function should return True.

Function 3: Stack2

Implement a class for stacks (called Stack2) using the std Queue class. In other words, the only operations that can be used in your Stack2 class are the ones defined in the Queue class.

Function 4: concatenate(llst1, llst2)

Given two (possibly empty) linked lists, concatenate them such that the first list comes first.

Example: When the first input linked list lst1 is $1 \to 2 \to 3$ and the second lst2 is $7 \to 8 \to 9$, the output linked list is $1 \to 2 \to 3 \to 7 \to 8 \to 9$.

Function 5: removeNodesFromBeginning(llst, n)

Remove the first *n* nodes from a (possible empty) linked list.

Note: $0 \le n \le lst.length()$

Example: Given an linked list $1 \to 2 \to 3 \to 4 \to 5$ and $n = 3$, return the linked list $4 \to 5$.

Function 6: removeNodes(llst, i, n)

Starting from the *i*th node in a (possibly empty) linked list, remove the next *n* nodes, not including the *i*th node.

- We say the head of the input list is the *first* node, i.e., $i = 1$.
- When $i = 0$, your function should start by removing the head.

Note:

- $n \ge 0$
- $i \ge 0$
- $i + n \le lst.length()$

Example: let lst be a linked list $1 \to 2 \to 3 \to 4 \to 5 \to 6$

1. When $i = 2$ and $n = 3$, return the linked list $1 \to 2 \to 6$.
   Explanation: The second node ($i = 2$) is 2. Removing the next three nodes ($n = 3$) means removing nodes 3, 4, and 5. The remaining ones are 1, 2, 6.
2. When $i = 0$ and $n = 3$, return the linked list $4 \to 5 \to 6$.

Explanation: Since *i* = 0, we remove the first three nodes (*n* = 3) from the beginning of lst, which are 1, 2, 3. The remaining ones are 4, 5, 6.

Function 7: buildHeap(arr)

Given an unsorted array of ints, apply the heap properties to the array and return it sorted in heap order.

For examples lst = `[7, 3, 20, 25, 12, 20, 4, 1, 9, 11]`

`Return arr = [1,3,4,7,11,20,20,25,9,12]`