Programming Assignment 1

Data Structures and Algorithms

BNU-HKBU United International College

Rubrics

- Refer to the Rubrics for programming on iSpace
- You will get full mark for Function test if
  - Your code produces correct output for all our test inputs (hidden).
    - The test inputs are not provided to you.
    - Try your code against all possible inputs (that you can think of) to test correctness
  - No memory leak is found in any case
- Program Structure refers to
  - Reasonable file structure in the project
  - Reasonable placement of function declarations and implementations
- Code style includes
  - Reasonable naming of identifiers
  - Reasonable indentation
  - Code neatness

Task1: List Methods

- Given the Linked list ADT introduced in Lecture 2, implement three more methods:
  - `removeNodes`
  - `removeDuplicates`
  - `reverseList`
- Complete List.java including
  - Class definition
  - Declaration and implementation for the existing and the new methods
    - Write your code based on the sample solution provided on iSpace.
  - A main function which runs your own test cases

removeNodes

- public int removeNodes(double x)
  - Removes all the elements from a linked list that have value *x*.
  - Returns the number of occurrences of *x.*
- Sample Input and output:

Input	**List Update**	**Returned Value**
2 -> 6 -> 5 -> 6 -> NULL, x = 6	2 -> 5-> NULL	2
NULL, x = 6	NULL	0
6 -> 6 -> 6 -> NULL, x = 6	NULL	3

removeDuplicates

- public void removeDuplicates()
  - Remove duplicates from sorted list (you may assume that the node values in the list are in non-decremental order)
  - Removes all nodes that have duplicate values, leaving only nodes with distinct values
  - Your implementation should finish the removing with SINGLE traversal of the whole list, which means you cannot simply invoke the removeNodes method for removing the duplicates.
- Sample Input and output:

Input	**List Update**
1 -> 2 -> 2 -> 4 -> 6 -> 6 -> NULL	1 -> 4-> NULL
NULL	NULL
6 -> 6 -> 6 -> 7 -> 7 -> NULL	NULL

reverseList

- public void ReverseList()
  - Reverse the linked list WITHOUT using extra space.
  - Hint: Reverse the linked list can be performed by modifying the “next” pointer of current node from the original next node to its previous node.
- Sample Input and output:

Input	**List Update**
1 -> 2 -> 3 -> 4 -> 5 ->NULL	5->4->3->2->1->NULL
NULL	NULL
1->NULL	1->NULL

Task 2: Postfix Expression Evaluation

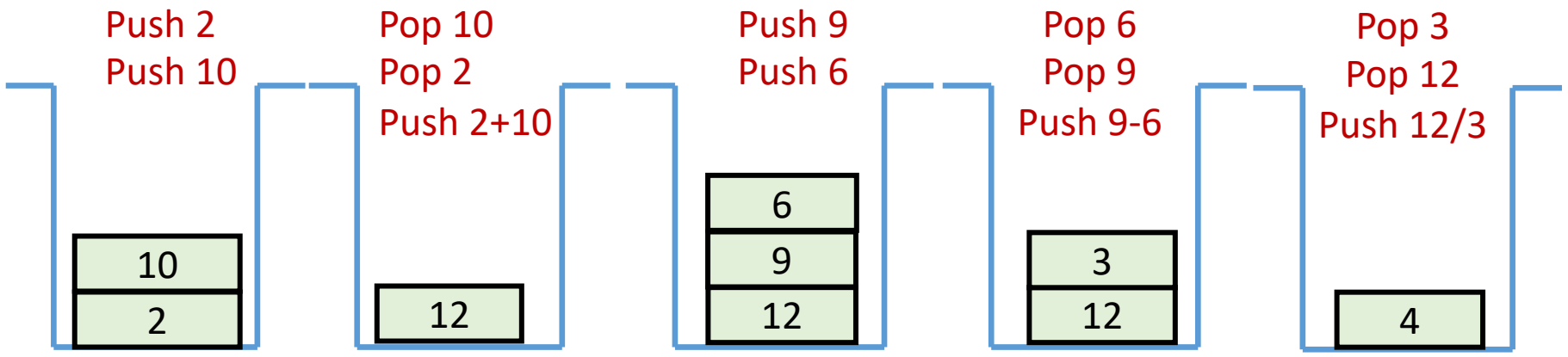
- Given the Stack ADT introduced in Lecture 3, implement one more static method:
  - `computePostfix`
- Complete Stack.java including
  - Class definition
  - Declaration and implementation for the existing and the new methods
    - Write your code based on the sample solution for the Stack class provided on iSpace.
    - You should modify the Stack class such that it manages objects of the Integer class.
  - A main function which runs your own test cases

Task 2: Postfix Expression Evaluation

- public static int computePostfix(String postfix)
  - postfix is a string representing a valid postfix expression which contains only digits and '+', '-', '*', '/' operators, separated by ','.
    - Hint: Use the split function for splitting a string by some delimiter
    - https://www.programiz.com/java-programming/library/string/split
  - You can assume the length of postfix will not exceed 50.
  - Returns the evaluation result.
- Consider the postfix expression evaluation algorithm introduced in Lec3:
  - If the element is an operand, push it to stack
  - If the element is an operator O, pop twice and get A and B respectively. Calculate BOA and push it back to stack
  - When the expression is ended, the value in the stack is the final answer.

Task 2: Postfix Expression Evaluation

- Sample input and output:
  - Postfix expression: 11,1,3,*,/ , result is 3, as 11/(1*3) = 3
  - Postfix expression: 2,10,+,9,6,-,/, result is 4 (as illustrated in the Figure).



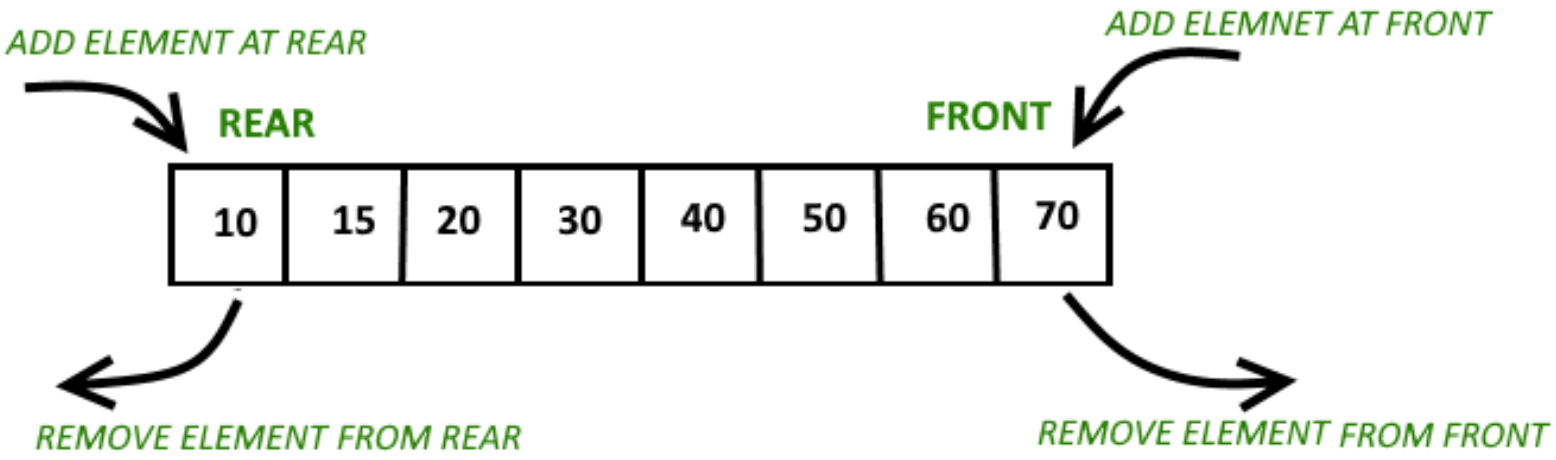
Task 3: Design Circular Deque

- Given the Queue ADT introduced in Lecture 4, implement the circular double-ended queue (deque)
- Deque or Double Ended Queue is a generalized version of the Queue data structure that allows insert and delete at both ends
- Complete CircularDeque.java including
  - Class definition
  - Declaration and implementation for the given methods
    - Write your code based on the sample solution for the Queue class provided on iSpace.
    - You should modify the Queue class such that it manages objects of the Integer class.
  - A main function which runs your own test cases

Task 3: Design Circular Deque

- Implement the CircularDeque class:
  - CircularDeque(int size) Initializes the deque with a maximum size of size.
  - Integer insertFront() Adds an item at the front of Deque. Returns this item if the operation is successful, or Null otherwise.
  - Integer insertLast() Adds an item at the rear of Deque. Returns this item if the operation is successful, or Null otherwise.
  - Integer deleteFront() Deletes an item from the front of Deque. Returns this item if the operation is successful, or Null otherwise.
  - Integer deleteLast() Deletes an item from the rear of Deque. Returns this item if the operation is successful, or Null otherwise.
  - boolean isEmpty() Returns true if the deque is empty, or false otherwise.
  - boolean isFull() Returns true if the deque is full, or false otherwise.
  - Void displayCircularDeque()

Task 3: Design Circular Deque



Submission

1. Submit the two java files to ispace:
   - List.java
   - Stack.java
   - CircularDeque.java
2. Submit them as two separate files. Don’t compress them!

Plagiarism Policy

- You are encouraged to collaborate in study groups.
  - But, you cannot copy or slightly change other students' solutions or codes.
- We will check between everyone's submission.
- We will check with online solutions.
- If copies are found, everyone involved gets ZERO mark.