Data Structures and Algorithms

Programming Assignment 1 Fall 2024

Department of Computer Science & Technology
United International College

Rubrics

Criteria for assessment	Performance levels				
	Excellent 10 / A / 4	Good 8 / B / 3	Satisfactory 6 / C / 2	Marginal Pass 4 / D / 1	Fail F / 0
Function test **(80 % weighting)**	All systems test case run successfully.	Most system test case run successfully.	Some unit test case runs successfully.	Only a few test cases successfully.	The code runs none.
Program structure **(10 % weighting)**	Needed program structures are evident.	Program structures are clear.	Program structures are obscure.	Needed program structures are lacking.	None.
Comment **(5 % weighting)**	Comments are adequately provided and are at levels of abstraction appropriate for conveying specifics about the programs.	Comments are mostly provided and at levels of abstraction appropriate for conveying specifics about the program.	Comments are provided somewhere, but at too low a level of abstraction to be of much use.	Comments are sparse or vague, and give little information about the purpose of the program or how it goes about carrying it out.	No comments and no information about the purpose of the program.
Code style **(5 % weighting)**	A clear coding style is evident, and consistently applied, greatly enhancing program readability	A clear coding style with mostly consistency in application, aiding readability in a majority of the program.	A clear coding style is hinted at, with some consistency in application, aiding readability in some of the program.	A clear coding style is lacking, or applied very inconsistency, with readability suffering accordingly.	None

Comments on the Rubrics

- You will get full mark for Function test if
  - Your code produces correct output for all our test inputs.
    - The test inputs are not provided to you.
    - Try your code against all possible inputs (that you can think of) to test correctness
- Program Structure refers to
  - Reasonable class structure in the project
  - Reasonable declarations and implementations for the methods
- Code style includes
  - Reasonable naming of identifiers
  - Reasonable indentation
  - Code neatness

PROBLEM LIST

Problem 1 – List Methods

- Given the Linked list ADT introduced in Lecture 2, implement two more methods:
  - `getLast`
  - `removeRange`
- 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 all the given sample inputs and outputs

getLast

- public Node getLast()
  - Returns (without removing) the last node in the list
  - Returns Null if the list is empty
- Sample Input and output

Input List	Returned Value
1 --> 8 --> 2 --> 4 --> 3	Node containing 3
null	null

RemoveRange

- public int RemoveRange(int start, int stop)
  - Deletes all the nodes whose positions are in range *[start, stop]* , both sides inclusive, and returns the number of nodes deleted.
  - The position of the head Node is 1.
  - If *start* is greater than *stop*, *[start, stop]* is empty, so no nodes should be deleted.

RemoveRange

- Sample Input and output

Input List	Start, Stop	List Update	Returned Value
1 --> 8 --> 2 --> 4 --> 3	1, 3	4 --> 3	3
1 --> 8 --> 2 --> 4 --> 3	-3, 3	4 --> 3	3
1 --> 8 --> 2 --> 4 --> 3	5, 9	1 --> 8 --> 2 --> 4	1
1 --> 8 --> 2 --> 4 --> 3	6, 9	1 --> 8 --> 2 --> 4 --> 3	0
1 --> 8 --> 2 --> 4 --> 3	-9, 9	null	5
1 --> 8 --> 2 --> 4 --> 3	5, 2	1 --> 8 --> 2 --> 4 --> 3	0
null	1, 4	null	0

Problem 2 – ValidBrackets

- Given the Stack ADT introduced in Lecture 3, implement one more static method:
  - validBrackets
- Complete stack.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
    - You should modify the stack class so that it stores chars
  - A main function which runs all the given sample inputs and outputs

Problem 2 – ValidBrackets

- public static boolean ValidBrackets(String str)
- *str* is a string containing only '(', ')', '{', '}', '[', ']', '<' and '>'
- The method returns *true* if the input string is valid and *false* otherwise
  - In a valid string,
    - The brackets must match
    - The brackets must close in the correct order
  - An empty string is valid
  - A null string is invalid

Problem 2 – ValidBrackets

- Sample Input and output

Input	Output
"{()<()>}[]"	True
"(<)>"	False
"{()}["	False
""	True
NULL	False

Problem 2 – ValidBrackets

- Hint
  - You can make use of the Stack ADT
  - Consider what action you will take when you process the following characters in the string
    - '{', '[', '(', '<': opening brackets
    - '}', ']', ')', '>': closing brackets

Submission

1. Submit the two java files to ispace:
   - List.java
   - Stack.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.