COSC 2100/2010 (Fall 2017)

Assignment III

Due by Monday, October 2, 2017 6:00 PM CST

**Submission Instructions (Please read):**

* Copy all source files (from both questions) into one a single folder named as **<GroupMember1\_Fullname>\_<GroupMember2\_FullName>\_Project3** then zip that folder and upload to D2L.
* Make sure to **put the full names of group members in the Java files.**
* One submission per group
* **Please use the folder naming format described above. Do not name folders such as src, project3, and myproject.**

**1.** Suppose you have two rail-roads modeled as linked list as shown in the figure below. The nodes are stations with names (A, B, etc.) and the links are direct path between the two stations. Write a complete Java program to find if these two rail-roads meet or not. If they happen to meet, then return the first common node where the meeting happens. If the rail-roads do not meet, then return null.

As shown in the figure below, the two rail-roads meet at station D since the 4th station in the first rail-road is same as the 3rd station in the second rail-road. So, station D should be returned.

Do not use Java’s built-in HashMap or any other classes from the Collections framework.



You can start with the following code:

// see class LLStringNode on page 106 for the full implementation of linked list node.

public class StationNode

{

private String info;

private StationNode link;

}

public class RailroadTest {

// Implement this method

public StationNode findFirstCommonNode(StationNode road1, StationNode road2) { }

}

In the main method of RailroadTest class, test findFirstCommonNode method with the following scenarios:

1. For cases when both railroads have same length or not.
   1. The railroads meet at the first station or last station.
   2. The railroads meet at an arbitrary position.
   3. Rail roads do not meet at all.

**2.** In the language Lisp, each of the four basic arithmetic operators appears before an arbitrary number of operands, which are separated by spaces. The resulting expression is enclosed in parentheses. The operations behave as follows.

* (+ a b c …) returns the sum of all operands, and (+) returns 0.
* (- a b c …) returns a-b-c- … and (- a) returns –a. The minus operator must have at least one operand.
* (\* a b c …) returns the product of all the operands, and (\*) returns 1.
* (/ a b c …) returns a/b/c/ … and (/ a) returns 1 / a. The divide operator must have at least one operand.

You can form larger arithmetic expressions by combining these basic expressions using a fully parenthesized prefix notation. For example, the following is a valid Lisp expression:

(+ (- 6) (\* 2 3 4) (/ (+ 3) (\*) (- 2 3 1)))

The expression is successively evaluated as follows:

(+ -6 24 (/ 3 1 -2))

(+ -6 24 -1.5)

16.5

Design and implement an algorithm that uses a stack to evaluate a legal Lisp expression composed of the four basic operators and integer values. Write a main program (LispEvaluator.java) that reads such an expression and demonstrates your algorithm.

**Sample inputs and outputs**

(+ 5 0 10) evaluates to 15.0

(+ 5 0 10 (- 7 2)) evaluates to 20.0

(+ (- 6) (\* 2 3 4) (/ (+ 3) (\*) (- 2 3 1))) evaluates to 16.5

(+ (- 632) (\* 21 3 4) (/ (+ 32) (\*) (- 21 3 1))) evaluates to -378.11764705882354

**Notes**

This assignment can be submitted by groups of 1 or 2. You may choose your own group member. If you need a group member, please contact the instructor. You may only choose a group member in your own section (9:00 AM or 10:00 AM) class. **Enroll in one of the available groups in the Assignment 3 group category on D2L.**

**Important Reminder**

This assignment can be done in groups of 1 or 2. All solutions must be group members’ own work. Academic honesty policy is strictly enforced in this class. Violators of this policy will be reported to the Provost's office. There are no exceptions. Please refer to the course syllabus for the academic honesty policy. If you have any questions about this policy, please contact the instructor.

**Grading (for each question)**

|  |  |
| --- | --- |
| **Grading Criterion** | **Points** |
| **Does not compile** | **0** |
| Compiles | 4 |
| Input, Output and Computation is valid | 26 |
| Test cases (The program demonstrates that it works under different test cases) | 10 |
| Coding Style (indentation, variable naming, etc.) | 5 |
| Documentation (comments are provided, author name is given at the top of the program) | 5 |