Lab08 – Recursion & Stack

Instructions: The purpose of this lab is to explore basic recursion and the use of a Stack.

PLEASE NOTE, ALL CLASSES MUST BE IN THE CSU.CSCI325 PACKAGE!

Part I:

While humans prefer infix notation of mathematical expressions, computers would prefer pre-fix or post-fix notation.

Infix: An expression is said to be in Infix notation if the operator appears in between the operands in the expression in the form of: operand1 operator operand2. For example, (A+B) * (C-D)

Prefix: An expression is said to be in prefix notation if the operator appears in the expression before the operands in the form of operator operand1 operand2. For example, *+AB-CD is equivalent to the expression shown above in infix notation.

Postfix: An expression is said to be in postfix notation if the operator appears in the expression after the operands in the form of operand1 operand2 operator. For example, AB+CD-* is also equivalent to the expressions shown above in infix and prefix notation.

1. Create a class called *ConvertExp*. This class should have a method called *isOperator* that takes a string with one character. *isOperator* should implement a **HashSet** to determine if the string passed in is one of the following operators { + , - , * , / } and return true if the string is an operator.
2. ***This part must be implemented using a Stack. If you do not use a stack and the autograder issues the points, the instructor will take the points off afterwards.***

   Add a method called *preToInfix* to *ConvertExp* that takes a String representation of a prefix expression and converts it into an equivalent Infix expression.

   For example:

   - Prefix Input : *+AB-CD
   - Infix Output : ((A+B)*(C-D))

   - Prefix Input : *-A/BC-/AKL
   - Infix Output : ((A-(B/C))*((A/K)-L))

   Algorithm for Prefix to Infix:

   - Read the Prefix expression in reverse order (from right to left)
   - If the symbol is an operand, then push it onto the Stack
   - If the symbol is an operator, then pop two operands from the Stack
   - Create a string by concatenating the two operands and the operator between them.
   - string = (operand1 + operator + operand2)
   - And push the resultant string back to Stack
   - (more on next page)

- Repeat the above steps until end of Prefix expression.
- If the prefix expression is malformed, return "Malformed expression: " + the prefix expression. For example, "Malformed expression: *****". (Hint: check for empty stack and/or catch EmptyStackException.)

See Javadoc for more information.

3. ***<u>This part must be implemented using a Stack. If you do not use a stack and the autograder issues the points, the instructor will take the points off afterwards.</u>***

   Add a method called *postToInfix* to *ConvertExp* that takes a String representation of a postfix expression and converts it into an an equivalent Infix expression.

   For example:

   - postfix Input : abc++
   - infix Output : (a + (b + c))

   - postfix Input : ab*c+
   - infix Output : ((a*b)+c)

   If the postfix expression is malformed, return "Malformed expression: " + the postfix expression. For example, "Malformed expression: *****". (Hint: check for empty stack and/or catch EmptyStackException.)

   (Hint: use the code from #2 above with a slight modification.)

Part II:

<u>This part must be implemented using a Stack. If you do not use a stack and the autograder issues the points, the instructor will take the points off afterwards.</u>

The kids in Mrs. Doubtfire's 3rd grade class would like to pass notes to each other while she is not looking. Since they do not have much time to write, they have devised an encoding scheme to write as few letters as possible. Their alphabet consists of only upper-case letters A to Z. Any time a character is repeated, they replace the repeated letters with one occurrence of the letter and the number of times it repeats.

For example:

- The message AAABBCDDDDDDE would be encoded as A3B2CD6E.
- The message XYZZZZZZZZZZQQ would be encoded as XYZ10Q2.

1. Create a class called *Compress* that has a method called *compress* that takes a String as input. The method should use a stack to process the string into a new encoded string as shown above.

If the method is called with a null or empty string, the method should return null. See Javadoc for more details.

Part III:

<u>This part must be implemented using Recursion. If you do not use Recursion and the autograder issues the points, the instructor will take the points off afterwards.</u>

A palindrome is a word, number, sentence, or verse that reads the same backward or forward. It derives from the Greek palin dromo, which means "running back again." In palindromes, spacing, punctuation, and capitalization are usually ignored. Below are some examples:

- racecar
- eye
- noon
- civic
- level
- Mom
- Anna
- Bob
- Elle
- Poor Dan is in a droop
- A nut for a jar of tuna
- A Santa at NASA
- A man, a plan, a cat, a ham, a yak, a yam, a hat, a canal-Panama!
- Are we not drawn onward, we few, drawn onward to new era?

1. Create a class called *Palindrome* that contains a method called *checkPal* that takes a String for input. The *checkPal* method will prepare a string by removing all spaces, punctuation, and case sensitivity. The *checkPal* method should call the *isIt* method with the prepared string. The *isIt* method should call itself recursively to determine if the string is a palindrome. Note that the autograder will not call *isIt* directly since the *isIt* method expects a string prepared as described. The autograder will use the *checkPal* method. Your instructor will expect the *isIt* method to be recursive.

   <u>Base case:</u>

   If the size of the string is 0 or 1, return true. It is a palindrome.

   <u>Recursive case:</u>

   If the first and last character of the string are not equal, return false. It is not a palindrome.

   Otherwise, make a recursive call with a (smaller) string without the first and last character of the string.

   See Javadoc.pdf for details.

Testing: You are responsible for manually testing your code. It is more efficient to test your code locally (i.e., using a main or driver class or test class) rather than pushing your code for each modification to cause the autograder to grade your code. However, you may push your code to see the autograder results as often as you like prior to the due date. Official grades will be taken from your grade.txt only after the due date has passed.

How to turn in:

Turn in via GitHub. Ensure the file(s) are in your lab08 directory and push via NetBeans or GitExt. (See in class discussions/PPT for directions.)

Due Date: Tuesday, April 17, 11:59pm.

Teamwork: No teamwork, your work must be your own.