CIS 3501 Data Structures

Program 1

Winter 2019

Due Dates: January 29 (Part 1)

February 7, 2019 (Parts 1 & 2, source code, input/output files)

The program will create and traverse expression trees in prefix, infix and postfix order. Print out traversal results and evaluate order of operations from prefix notation.

The input will be read from a text file and each line will contain one infix expression to be processed. The requirements for the infix expression are:

* single uppercase letter operands
* operators are +, -, \* and / with \* and / having higher precedence than + and –
* normal left to right precedence for operator at same level
* a minimum of one space on each side of am operand (including initial and final operand)
* a minimum of one space on each side of each operator
* a minimum of one space on each side of parenthesis (including initial and final parenthesis)

Examples (MUST be included with program turn-in test data):

* ( ( ( A + B ) ) ) ) )
* ( A + B \* C ) / ( ( A – B ) \* C )
* X \* Y + A
* ( A + B ) / C + (D – E ) \* F \* ( G – H )
* A + B / ( ( C + D ) \* A – G )
* A + b – ) D / E ( F + 4 )

The program will have the user enter the name of the file - need to be able to handle exceptions such as file errors (no input file exists, input file empty, etc.), invalid data (invalid operand, invalid operator, spacing, mismatched parenthesis, etc.), etc. The name of the file with statements in it is named “statements.dat” – you will upload all input and output files to canvas. Note: empty files cannot be uploaded to canvas.

For each input line, the program will:

1. Check input line for errors
   1. print ALL errors detected
   2. if any errors detected go to next input line
2. Convert infix statement to postfix notation and list expected output operations
3. Build the corresponding expression tree and print the tree
4. Print the traversal of the expression tree in prefix, infix and postfix order. Note: infix would be incorrect for evaluation as it will not have parenthesis (which is why infix changed to postfix in step 1)
5. Evaluate value of expression using prefix notation
   1. Division will be integer division
   2. A = 1, B = 2, C = 3 …..
   3. What if you encounter a division by zero condition?

The output for the program will be written to the screen and to a file named output.dat which will contain some or all of the following (e.g. if input file cannot be opened there is obviously only the program completed message after file open error message):

* program started message
* files open success or fail message
* echo print input line
* if invalid statement, print errors including the piece(s) of data that caused error if appropriate (e.g. invalid operator or operand)
* print postfix expression and expected order of operations in postfix
* print expression tree
* print prefix, infix, and postfix traversals
* print individual operations to obtain expression result and final result
* program complete message

Sample operation output:

Input Line: A + B

Valid Statement

Postfix: AB+

Operations:

AB+

Expression Tree:

+

A B

Prefix: +AB

Infix: A+B

Postfix: AB+

Evaluation:

A + B = 3

Final Result: 3

Input Line: ( ( ( A + B ) ) ) ) )

Invalid Statement:

Mismatched Parenthesis

Input Line: ( A + B ) \* ( C – D )

Valid Statement

Postfix: AB+CD-\*

Operations:

AB+

CD–

AB+CD-\*

Expression Tree:

\*

+ -

A B C D

Prefix: \*+AB-CD

Infix: A+B\*C-D (remember no parenthesis needed since not stored)

Postfix: AB+CD-\*

Evaluation:

+AB = 3

-CD = -1

\*+AB-CD = -3

Final Result: -3

You may assume each component of the input will be a single character in size.

Your program may be written in any programming language, however, you may not use any built in methods other than edit methods. That is, you may use built in method isDigit but you cannot use built in tree class or built in expression converters.

Use turn-in template for document submission.