# Infix to Postfix Assignment

## Assignment objective:

Implement a dynamic array stack data structure and use it to implement the infix arithmetic expression to post fix arithmetic expression conversion algorithm.

## Overview:

The arithmetic expressions given as input to your program contain operands which are positive integers or single letter variables. The arithmetic expressions given as input to your program can contain any of the operators: +, – , *, / as well as parenthesis. The operators have the usual precedence as given in the table below. Parenthesis have the highest precedence. The operators * and / have the next lowest precedence. The operators + and – have the lowest precedence.

| Precedence | Operator | Type | Associativity |
|---|---|---|---|
| 1 | ( ) | Parenthesis | Left to Right |
| 2 | *<br>/ | Multiplication<br>Division | Left to Right |
| 3 | +<br>– | Addition<br>Subtraction | Left to Right |

Here are some examples of expressions written in infix notation and their postfix notation equivalent:

A simple infix expression: a + 5
Written as a postfix expression: a 5 +

A more complex infix expression: a + 156 * b
Written as a postfix expression: a 156 b * +

Another infix expression: a * 5 + b
Written as a postfix expression: a 5 * b +

Another infix expression: a + b * c – d
Written as a postfix expression: a b c * + d –

Another infix expression: a – ( b + c ) * d
Written as a postfix expression: a b c + d * –

## Infix to Postfix Algorithm:

You'll use a stack to process the operators and parenthesis in the arithmetic expression. Treat the arithmetic expression as a linear sequence of tokens. A token is a String that is an operand (positive integer or variable), an operator, or a left or right parenthesis.

The data member **ops** is a stack of Strings, the data member **postfix** is a queue of Strings, and the parameter **lineTokens** is a queue of Strings.

While **lineTokens** is not empty:

1. Dequeue a token from **lineTokens** and get its first character.
2. If the first character is a letter or digit, then the token is an **Operand token**. Enqueue the token onto the queue **postfix**.
3. If the first character is a left parenthesis, then the token is a **Left Parenthesis token**. Push the token onto the top of the stack **ops**.
4. If the first character is a right parenthesis, then the token is a **Right Parenthesis token**.
   a. Pop the operator at the top of the stack **ops** and enqueue the popped operator onto the queue **postfix**.
   b. Repeat step **a** above until the top of the stack **ops** contains a left parenthesis.
   c. Pop the left parenthesis at the top of the stack **ops** and discard it.
5. If the first character is one of the 4 operators: +, –, *, or /, then the token is an **Operator token**.
   a. If the stack **ops** is not empty and the operator at the top of the stack **ops** has a higher or equal precedence than the operator token just read, then pop the operator at the top of the stack **ops** and enqueue the popped operator onto the queue **postfix**.
   b. Repeat step **a** above until the stack **ops** is empty or the operator at the top of the stack **ops** has a lower precedence than the operator token just read.
   c. Finally, push the operator token just read onto the top of the stack **ops**.

   **Note:** Use the given method **hasHigherOrEqualPrecedence** to compare the precedence of two operators to see if the first operator has a higher precedence than the second operator.

After the last token in the arithmetic expression is dequeued from the queue **lineTokens** and processed, pop any remaining operators from the top of the stack **ops** and enqueue the popped operator onto the queue **postfix** until the stack **ops** is empty.

## Design:

1. Download the files **Assignment2.java**, **DynamicArrayStack.java**, **InfixToPostfix.java**, **InputReader.java**, **LinkedQueue.java**, **Queue.java**, **Stack.java**, and **TokenizeLine.java** provided along with this assignment in Blackboard and include them in your project.

2. You **cannot** modify any of the code in the files **InputReader.java**, **LinkedQueue.java**, **Queue.java**, and **TokenizeLine.java**.

3. The file **Stack.java** contains the stack interface. You **cannot** modify any of the code in this file.

4. The file **DynamicArrayStack.java** implements a stack using an array as storage where the stack's array grows and shrinks in size as discussed in class. You **cannot** change any of the code that is already in this file. You will write the code for the **size**, **resize**, **push**, **top**, and **pop** methods.

5. The file **InfixToPostfix.java** implements the infix to postfix conversion algorithm. You **cannot** change any of the code that is already in this file. You will write the code for the **convertInfixToPostfix** method to implement the infix to postfix algorithm as described on the previous page. You can add any additional methods to this class as needed but you must declare them **private**. You cannot and do not need to add any additional data members to this class.

6. The file **Assignment2.java** is the driver code for the assignment. You **cannot** and **do not** need to modify any of the code in this file. The main function receives, via the command line argument, the name of a text file containing lines of text. The main method creates an InputReader object and then calls the readAndReturnFileLines method of the InputReader object. The readAndReturnFileLines method returns a LinkedQueue of Strings containing the lines of text from the text file. The main method then dequeues each line of text from the LinkedQueue and passes the line of text to the generateTokenQueueFromLine method of the TokenizeLine class to create a LinkedQueue of Strings with each String in this queue being: a variable, a positive integer, an operator, or a left or right parenthesis. The main method finally passes this LinkedQueue to the convertInfixToPostfix method of the InfixToPostfix class which will convert the infix expression to its equivalent postfix expression returning the postfix expression back to the main method as a String to print.

7. Each line in the test file is an arithmetic expression written in infix notation.

8. The command to launch your program, assuming **test.txt** is the name of the text file, is:
   ```
   java Assignment2 test.txt
   ```

9. Do **NOT** use packages in your program. If you see the keyword **package** on the top line of any of your .java files then you created a package.

10. Do **NOT** use any graphical user interface code in your program!

11. Do **NOT** type any comments in your program. If you do a good job of programming it will be easy for me to determine the task of your code.

## Grading Criteria:

The assignment is worth a total of 20 points, broken down as follows:

1. If your code does not implement the task described in this assignment, then the grade for the assignment is zero.
2. If your program does not compile successfully then the grade for the assignment is zero.
3. If your program produces runtime errors which prevents the grader from determining if your code works properly then the grade for the assignment is zero.

If the program compiles successfully and executes without significant runtime errors, then the grade computes as follows:
Followed proper submission instructions, 3 points:

1. Was the file submitted a zip file?
2. The zip file has the correct filename.
3. The contents of the zip file are in the correct format.

Code implementation and Program execution, 12 points:

- The code performs all the tasks as described in the assignment description.
- The code is free from logical errors.
- The sparse matrix operations have the appropriate run-time.
- Program input, the program properly processes the input.
- Program output, the program produces the proper results for the assignment.

Code readability, 5 points:

- Good variable, method, and class names.
- Variables, classes, and methods that have a single small purpose.
- Consistent indentation and formatting style.
- Reduction of the nesting level in code.

**Late submission penalty:** assignments submitted after the due date are subjected to a 2-point deduction for each day late.

**Late submission policy:** you **CAN** submit your assignment early, before the due date. You are given plenty of time to complete the assignment well before the due date. Therefore, I do **NOT** accept any reason for not counting late points if you decide to wait until the due date (and the last possible moment) to submit your assignment and something happens to cause you to submit your assignment late. I only use the date submitted, ignoring the time as well as Blackboard's late submission label.

## Submission Instructions:

Go to the folder containing your **DynamicArrayStack.java** and **InfixToPostfix.java** files, select them, and place only them in a Zip file. The file can **NOT** be a **7z** or **rar** file! Follow the directions below for creating a zip file depending on the operating system running on the computer containing your assignment's two java code files.

Creating a Zip file in Microsoft Windows (any version):

1. Right-click any of the two files to display a pop-up menu.
2. Click on **Send to**.
3. Click on **Compressed (zipped) Folder**.
4. Rename your Zip file as described below.
5. Follow the directions below to submit your assignment.

Creating a Zip file in MacOS:

1. Click **File** on the menu bar.
2. Click on **Compress**.
3. MacOS creates the file **Archive.zip**.
4. Rename **Archive** as described below.
5. Follow the directions below to submit your assignment.

Save the Zip file with the filename having the following format:
your last name,
followed by an underscore _,
followed by your first name,
followed by an underscore _,
followed by the word **Assignment2**.
For example, if your name is John Doe then the filename would be: **Doe_John_Assignment2**

Once you submit your assignment you will not be able to resubmit it!
Make absolutely sure the assignment you want to submit is the assignment you want graded.
There will be **NO** exceptions to this rule!

You will submit your Zip file via your CUNY Blackboard account.
The only accepted submission method!

Follow these instructions:
Log onto your CUNY Blackboard account.
Click on the CSCI 313 course link in the list of courses you are taking this semester.
Click on the **Assignments** tab in the red area on the left side of the webpage.
You will see the **Infix to Postfix Assignment**.
Click on the assignment.
Upload your Zip file and then click the submit button to submit your assignment.

**Due Date:** Submit this assignment on or before 11:59 p.m. Thursday, May 2, 2024.