Points: 100

Due Date: Friday, March 22nd @ 11:59pm

This is an individual assignment.

**As usual you cannot import any classes from the Java Library.**

**You must use your (i.e. from the textbook) implementations of the Java classes used for this assignment.**

**Exceptions:**

**You may now use (and import if necessary):**

* **List**
* **Iterable**
* **Iterator**
* **ArrayList**
* **String**
* **Scanner**
* **Random**
* **File**
* **JOptionPane**
* **You may also import any needed Exception classes.**

**Task:**

Write a Java project that:

* Implements the Linked Binary Tree ADT presented in the textbook (i.e. your Lab107).
* Implements the Shunting Yard algorithm to convert an in-fix expression into its corresponding post-fix notation.
* Uses the queue and stack approach to evaluate an expression that is in post-fix notation.
* Uses the queue and stack approach to build the binary expression tree for an expression that is in post-fix notation.

You must use your implementations of stacks and queues from previous assignments.

Your program must:

* Ask the user to enter the absolute path and filename (as a single String) of the file that contains a list of arithmetic expressions. Each expression will be on a single line in the input text file delimited by an end of line character.
* Read arithmetic expressions from an input file until the EOF is reached.
  + See file format and example at end of assignment.
* For each expression your program should:
  + Print out the expression that was read from the file.
  + Determine if the expression is valid.
  + For each invalid expression:
    - Print an invalid expression message.
    - Your messages does NOT need to explain why the expression is invalid.
  + For each valid expression:
    - Evaluate the expression and display the results of the evaluation
      * You must assume that all numbers are floating point numbers
    - Print the expression in post-fix notation
    - Represent the expression as a binary expression tree. The binary expression tree should not be printed.
    - Print the pre-order traversal of the expression tree
    - Print the in-order traversal of the expression tree
    - Print the post-order traversal of the expression tree
    - Print the expression using Euler’s tour so that it produces a traditional parenthesized expression.

**Input file format:**

Each token in the input file will be blank separated so the expressions should be easy to parse.

Tokens will be one of the following:

* Numeric values (possibly includes negative numbers and decimal places):
  + The uniary negative operator will not have a blank space between the operator and its corresponding operand, e.g. -45
  + The binary subtraction operator will have blank space between the operator and its corresponding operands, e.g. 11 - 5
* Operators will be limited to:
  + Addition +
  + Subtraction -
  + Multiplication \*
  + Division /
* Parenthesis
  + In order to make expression more readable parenthesis, curly brackets and square brackets may be used.
  + For grouping and nesting purposes the symbols must match correctly.
  + For example:
    - ( 3 - [ { 4 / 3 } + 7 ] - 2 ) is correct nesting
    - ( { [ } ] ) is incorrect nesting
* There will be no “implied” multiplication
  + The expression 3 \* ( 4 - -5.0 ) is valid
  + The expression 3 ( 4 - -5.0 ) is not valid
* You do **not** need to check for invalid tokens.

**Turning in your assignment:**

* **Make sure that all of your code is properly documented.**
* Turn in your assignment using the standard method.
* Copy and paste each of your Java files into the document.
* Paste the screenshots showing the complete output of a complete run of your program after the Java code in your document.
* Export your NetBeans project to a zip archive.
* Turn in the Word document and zipped project as to separate files in a single Blackboard submission.
* You do not need to turn in your data files. We will test your program with a standard set of test files.

**Example input file ( data.txt ):**

**Caution: Do NOT cut and paste the example data from this Word document. You may pick up some "unprintable" ASCII characters when you cut code from a file that is stored in a binary (non-ASCII) format.  A Word document may contain formatting format information (encoded as non-printing characters) that is not visible in an ASCII editor like Notepad or the editor used in NetBeans.**

**These examples should not be considered an exhaustive test of your algorithm.**

**It is your responsibility to develop a set of test expressions that completely tests your algorithm.**

3 \* -5

4.5 – 3.6 / 5.2

( 4 - 3 ) / 5

4 + ( 7 / 2 )

[ 4 + 7 ] \* { 8 - 11 }

4 + 7 8 - 11

( ( [ 3 + 1 ] \* 3 ) / ( ( 9 - 5 ) ) - ( ( 3 \* ( 7 - 4 ) ) + 6 ) )

( ( 3 + 1 ) \* 3 ) / ( ( 9 - 5 ) ) - ( ( 3 \* ( 7 - 4 ) ) + 6 ) )

3 + 1 \* 3 / 9 - 5 - 3 \* 7 - 4 + 6

42

8 \* 24 / ( 4 + 3

3 + 4 –