CMPE 242
Spring 2020
Programming Homework #3

This assignment is due by 23:55 on Sunday, 10 May 2020 .

Late submissions: 20 points will be deducted for each late day.

Note: This assignment will not be evaluated without a face-to-face demo.

In this assignment, you will get familiar with an important programming concept, i.e., algebraic expression tree and different expression notations such as *prefix*, *postfix*, and *infix*. You will be using the concepts of binary tree data structure to convert and evaluate expressions.

ALGEBRAIC EXPRESSIONS

An algebraic expression can be represented using three different notations:

- **Infix**: The most conventional way to write an expression is called infix notation. The arithmetic operators appear in between two operands. e.g., 2+5*3+1
- **Prefix**: In prefix notation, as the name suggests, operators come before the operands. e.g., + + 2 * 531.
- **Postfix**: In postfix notation, different from infix and prefix notations, operators come after the operands. e.g., 253 * +1+

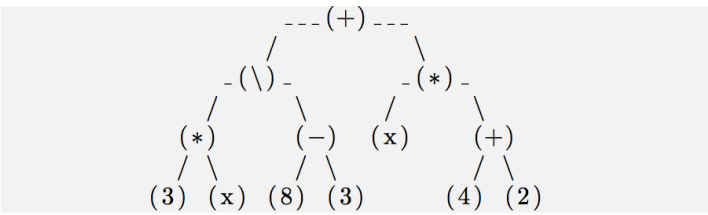
If you do not feel comfortable with these concepts, please check your course textbook.

ALGEBRAIC EXPRESSION TREES

One use of the binary trees in computer science is to represent algebraic expressions. In an algebraic expression tree, each leaf node contains an operand and a non-leaf node, including the root node, contains an operator that is applied to the results of its left and right sub-trees. For example, the expression below:

```
3*x/(8-3)+x*(4+2)
```

can be represented as



QUESTION

In this assignment, you are supposed to implement an algebraic expression tree using the binary tree data structure. The data structure will represent a single arithmetic expression, which will be given into the constructor with a String parameter. In the constructor, this parameter will be parsed and a binary tree will be constructed. Then, a user will be able to perform tasks on the constructed tree, including expression evaluation, expression conversion, and tree display.

In your implementation, you are supposed to write a separate method for each of these tasks. The details of these functions (tasks) are given below:

1. `AlgExpressionTree(String expression)`

This is the constructor that takes a String containing the expression in the prefix notation and parses this expression in order to construct its corresponding algebraic tree.

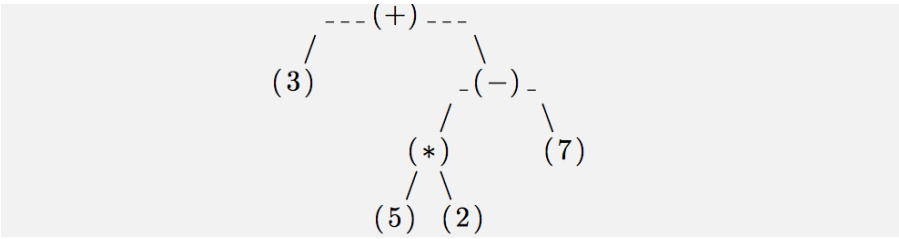
For simplicity, you may assume the following:

- An expression contains variables with single-character names (such as *x*, *y*, and *z*) and single-digit integers (0, ..., 9). However, in the expression, there may exist multiple occurrences of the same or different variable names and multiple occurrences of the same or different integers.
- An expression contains the basic four operators, which are `+,-,*,/`
- In an expression, no whitespace is allowed between the operator and operands.

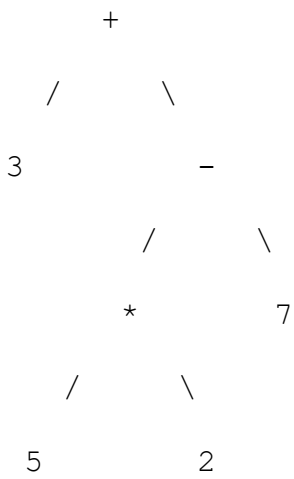
Please note that once you have obtained the algebraic tree for a given expression, the other tasks, which are explained below, become just a matter of traversing this tree. Thus, please be sure to implement this constructor correctly.

2. `void displayTree()`

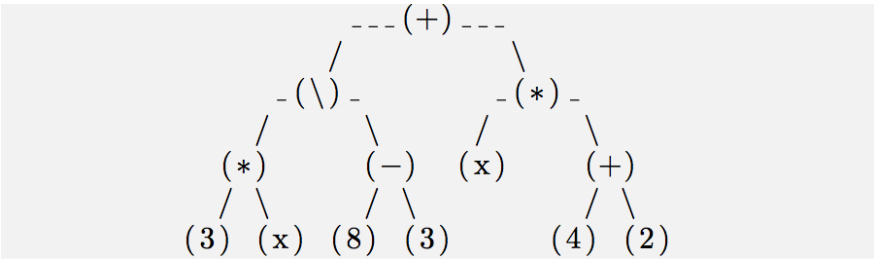
This function displays the tree in a graphical format as given in the example output. The function should write every node at the same level into the same column. That is the output of the `displayTree` function will be the 90 degree rotated version of the original tree display without lines between keys. For example, a tree that is constructed from an expression $3 + (5 * 2 - 7)$ which has a prefix notation $+3 -* 527$ will originally have a display given below.

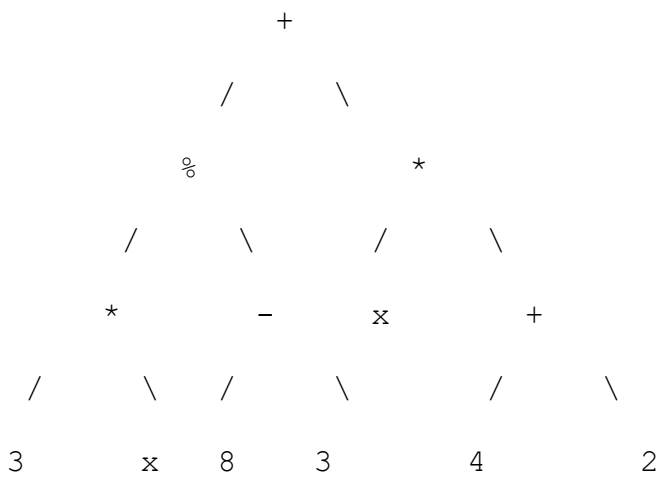


i.e. the output of the 'displayTree' function should be as follows:



Similarly, the original display and the output of a tree constructed from an expression $(8 - 4/2) + 3 * 4$ which has a prefix notation $+ - 8/42 * 34$ is given below as another example.





HINT: create a 2D character array that is large enough to store the table output, organize it as a grid, and then write the variables, numbers, and the links based on the level of the nodes in the tree.

3. void displayPostfix()

This function displays the *postfix* notation of the expression represented by the algebraic tree.

IMPORTANT NOTE: In your program, you should make this con- version on the algebraic expression tree. If you do it on the corresponding expression string, you will get NO POINTS for this part.

4. void setVariable(char varName, int varValue)

This function assigns an integer value *varValue* to the variable whose name is *varName*. If the variable with this name does not exist in the expression, this function will not make any assignment.

Note that this function does NOT permanently assign a value to the given variable. Thus, the value of any variable can change many times during the program, calling this function repeatedly. See the example program and output below for better understanding of how this function works.

5. double evaluate()

This function evaluates the expression represented by its algebraic tree and returns the result. For each particular variable, it uses the integer value assigned by the most recent call of the *setVariable* function for this particular variable. If the *setVariable* function has not been called for a variable, it uses the default integer value of 0. Again see the example program and output below for better understanding of how this function works.

IMPORTANT NOTE: In your program, you should make these evaluations on the algebraic expression trees. If you evaluate them on the corresponding expression strings, you will get NO POINTS for this part.

Below is the required public part of the **AlgExpressionTree** class that you are going to implement for this assignment. The name of the class MUST be **AlgExpressionTree**, and it MUST include the following public methods. We will use these functions to test your code. In your implementation, you may define additional public and private member functions and data members of this class. You may also define additional classes in your solution.

```
class AlgExpressionTree {
public:
     AlgExpressionTree(String expression);
     void displayTree();
     void displayPostfix();
     void setVariable(char varName, int varValue);
     double evaluate();
}
```

SAMPLE PROGRAM AND OUTPUT

```
public static void main( String[] args) {

       AlgExpressionTree T1 = new AlgExpressionTree("+*3x5");
       AlgExpressionTree T2 = new AlgExpressionTree("++y9z");
```

```
        T1.displayTree();

        // Evaluate the expression for x = 0 (default value):
        System.out.println("Result for x = 0   : " + T1.evaluate());

        // Evaluate the expression for x = 8, but do not change the tree:
        T1.setVariable('x', 8);
        T1.displayTree();
        System.out.println("Result for x = 8   : " + T1.evaluate());

        // Evaluate the expression for x = 6, but do not change the tree:
        T1.setVariable('x', 6);
        System.out.println("Result for x = 6   : " + T1.evaluate());

        // The setVariable function does not do anything
        // since y does not exist in the expression and
        // the evaluate function uses the most recent value of x
        T1.setVariable('y', 10);
        System.out.println("Result for y = 10   : " + T1.evaluate());

        // Print the tree in postfix form
        System.out.println("Postfix form: ");
        T1.displayPostfix();

        // Evaluate the expression for z = 10
        // Note: uses the default value for y, which is 0
        T2.setVariable('z', 10);
        System.out.println("Result for z = 10   : " + T1.evaluate());

        // Evaluates the expression for z = 10 and y = 3
        T2.setVariable('y', 3);
        System.out.println("Result for y = 3 : " + T2.evaluate());

        return 0;
}
```

OUTPUT:

```
          +
        /     \
      *        5
   /     \
3         x
Result for x = 0   : 5
          +
        /     \
      *        5
   /     \
3         x
Result for x = 8   : 29
Result for x = 6   : 23
Result for y = 10   : 23
Postfix form: 3x*5+
Result for z = 10  : 19
Result for y = 3   : 22
```

IMPORTANT

IMPORTANT NOTES: Do not start your homework before reading these notes!!!

1. You should submit your homework to course moodle page before the deadline. No hardcopy submission is needed. You should send files and any additional files if you wrote additional classes in your solution as a single archive file (e.g., zip, rar).

2. The below rules about late homework submissions apply. Please see the course syllabus for further discussion of the late homework policy as well as academic integrity.

3. You ARE NOT ALLOWED to modify the given method names. However, if necessary, you may define additional data members and member functions.

4. For this assignment, you are allowed to use the codes given in our textbook and/or our lecture slides. You ARE NOT ALLOWED to use any codes from somewhere else (e.g., from the internet, other text books, other slides ...).

5. The submissions that do not obey these rules will not be graded.

6. To increase the efficiency of the grading process as well as the readability of your code, you have to follow the following instructions about the format and general layout of your program.

7. Do not forget to write down your id, name, section, assignment number or any other information relevant to your program in the beginning of your Java files. Example:

```
//-------------------------------------------------------
// Title: BST
// Author: Hamid AHMADLOUEI
// ID: 2100000000
// Section: 0
// Assignment: 1
// Description: This class defines a BST
//-------------------------------------------------------
```

8. Since your codes 1will be checked without your observation, you should report everything about your implementation. Well comment your classes, functions, declarations etc. Make sure that you explain each function in the beginning of your function structure. Example:

```
void setVariable(char varName, int varValue)
//-------------------------------------------------------
// Summary: Assigns a value to the variable whose
// name is given.
// Precondition: varName is a char and varValue is an
// integer
// Postcondition: The value of the variable is set.
```

```
//-------------------------------------------------------
{
    // body of the function
}
```

- Indentation, indentation, indentation...

- This homework will be graded by your TAs, Hamid Ahmadlouei and İbrahim İleri (hamid.ahmadlouei@tedu.edu.tr, ibrahim.ileri@tedu.edu.tr). Thus, you may ask them your homework related questions. You are also welcome to ask your course instructor Tolga Çapın and Tansel Dökeroğlu for help.

GRADING RUBRIC

Performance Element	Weight	Master (4/4)	Advanced (3/4)	Developing (2/4)	Beginner (1/4)	Insufficient (0/4)
Information	5%	Information (id, name, section, assignment number) given in each file.	Missing minor details.	Missing few details (e.g. section id).	Only name given.	None.
Documentation	5%	Every class and method has a detailed explanation (pre- and post-conditions, sample I/O, etc).	Most classes and methods have an explanation, but missing in some parts.	Attempted to document the classes and methods, but they are not clear.	Only few comments.	Not even an attempt.
Code Design	5%	Modular source code and code format. Complete submission of classes and methods.	Methods make sense. Includes constructor that initializes carefully.	Uses set/get methods as necessary.	Class does very little; most functions remain in one main (driver) class.	Methods not properly defined.
Abstract Data Type: Binary Tree	10%	Use binary tree based data structure implementation. And provides all functionality.	Uses binary tree data structure for implementation. And provides most of expected functionalities.	Implements tree with major deviations from the specification.	Used different data structure from tree to solve this problem.	Not even an attempt.
Functionality	65%	All functions are implemented with no missing functionality. Runs without any crash.	Missing some minor features or minor output problems. Runs without any crash.	Attempted to implement all functions but some of them do not work.	Only few functions are implemented correctly. Compiles but several warnings.	No working solution or does not compile.
Testing: Test data creation & generation	10%	Provided a tester class. Able to identify key test points. Clear on what	Provided a tester class. Able to identify key test points. Clear on what aspects of the	Provided a tester class. Able to identify key test points.	Sporadic.	No test case presented.

		aspects of the solution are being tested with each set. Able to generate test data automatically when necessary.	solution are being tested with each set.			