BIL 212:Summer 2021

Assignment 2

Assigned on June 25. You should upload your program to uzak.etu.edu.tr by Wednesday, July 7 (any time up to midnight). When uploading do not compress your source directory, upload each file in your source directory separately, and do NOT use packages. Remember that your code should be fully documented. I once again remind you to read the academic honesty policy stated in the syllabus.

Overview: The goal of this assignment is to implement an **ExpressionTree** class that represents an arithmetic expression as discussed in class. The leaf nodes store operands and the internal nodes store operators. Binary operators +, -, *, and unary operators sin and cos are allowed. The operands are either constant numbers (you may assume they are integers) or the variable X.

Your implementation MUST follow the specifications given below:

- The ExpressionTree class MUST extend the LinkedBinaryTree class we studied in class.
- The constructor of the ExpressionTree class should take a String that contains a fully parenthesized infix expression and construct the corresponding tree. For example $(X + (5 * 7))$ is a fully parenthesized input where as $4 + 5 * 7$ is not. For simplicity you may assume the parenthesis, the operands and the operators are separated by white space in the input String. $((cos(2 * X)) + (5 * 7))$ is an example including a cos.
- In ExpressionTree class, write a toString() method that would return a String containing the expression in infix form. Note that the infix form has to be fully parenthesized. For unary operators, operator appears before operand, e.g. cos X
- In ExpressionTree class, write a method **int evaluate(int xvalue)** that would evaluate the expression stored in the tree for the given value xvalue of the variable X, and return the result.
- In ExpressionTree class, provide a method **ExpressionTree derivative()** that creates and returns a new ExpressionTree which is the derivative of this expression tree with respect to variable X. This new tree should be storing the derivative of the expression in its simplest form (see below for an example.) Implement this method using recursion and the following rules:
  - Derivative of a constant is 0
  - Derivative of X is 1: Derivative(X) = 1
  - The derivative of the sum of expressions S and T is the sum of the derivative of expression S and the derivative of expression T, that is, Derivative(S + T) = Derivative(S) + Derivative(T)
  - The derivative of the difference between expressions S and T is the derivative of expression S minus the derivative of expression T, that is Derivative(S - T) = Derivative(S) - Derivative(T)

- Product rule: Derivative(S * T) = (S * Derivative(T)) + (T * Derivative(S))
- Derivative(cos (S)) = 0 - (sin (S) * Derivative(S))
- Derivative(sin (S)) = cos (S) * Derivative(S)

Note that, the derivative you compute recursively may not be in its simplest form. So you have to simplify it. For example, if the expression is ((3 * X) - 5),

```
Derivative(((3 * X) - 5)) = (Derivative((3 * X)) - Derivative(5))
                          = (((3 * Derivative(X)) + (X * Derivative(3))) - Derivative(5))
                          = (((3 * Derivative(X)) + (X * 0)) - 0)
                          = (((3 * 1) + (X * 0)) - 0)
                          = 3                          //After Simplification
```

I suggest first creating the derivative tree ignoring simplification and then postprocess the tree to perform simplification.

In order to do the simplification write a method **ExpressionTree simplify()**. This returns a new expression tree which is (possibly) simpler than the original one. The original tree is not changed. Use the following two simplification rules:

- Whenever a and b are constants (integers), replace subexpressions of the form (a + b), (a - b), (a * b) by their integer values.
- If x is a subexpression, replace (x + 0), (0 + x), (x - 0), (x * 1), and (1 * x) by x. Similarly, replace (x * 0) and (0 * x) by 0.

- In ExpressionTree class, provide a method **displayTree()** that would display the tree as follows:
  - Nodes should be printed according to their order in an *inorder* traversal, one node per line.
  - The data for a node should be preceded by *depth* number of dots, where *depth* denotes the depth of the node in the tree.

  Here is an example output for the tree corresponding to $(4 + (5 * 7))$.

```
. 4
+
. . 5
.*
. . 7
```

- Write a driver program to test all methods of the `ExpressionTree` class.