CENG 202: DATA STRUCTURES

Programming Assignment 4

Deadline: WEDNESDAY, 25 MAY 2023 by **20:00**

Rules

1. You will use Java to implement the tasks
2. You will lose 20 points for each day your assignment is submitted late.
3. No deadline extensions will be granted.
4. Your code must be developed individually. You cannot collaborate with your peers. You cannot post your code on any public repository, nor can you share it with anybody. Any violations to this code of conduct will be treated as academic misconduct.
5. Submitted code will be automatically checked using tools that detect plagiarism.
6. In case cheating is detected, the grade of this assignment will be zero and a further penalty of 50 points will be applied to the next homework assignment.
7. You may be asked to explain your code and rewrite some parts of it in front of the instructor.
8. Submit your codes to AYBUZEM.
9. Write your name as a comment line in each file you submit.
10. In order to be accepted, your submission
    - must be of text format with a '.java' extension (no Word or other formats and no archives will be accepted)
    - You will receive a zero grade if you do not submit the assignment or you do not adhere to this rule!

Note: You can only use code/pseudo code from the slides and the textbook. You need to develop the rest of the code yourself.

What to turn in electronically:

- Your source code (*.java file).

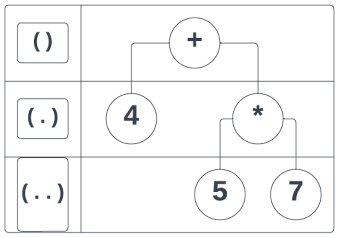
Task:

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 whereas **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 returns 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 evaluates the expression stored in the tree for the given value of the variable **X**, and returns the result.
- In `ExpressionTree` class, provide a method `displayTree()` that displays 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.