CSCI 241, Summer 2013

Assignment #1: Expression Trees (30pts)

Summary

- Goal: This assignment is worth 15% of the grade for the course. In this assignment you will implement some functions for building and traversing trees. Please work with your partner(s) (1-3 people per group) in completing this assignment, and continue to follow good design practices!
- Due time: **end of day (11:59pm), Thursday, July 11.**
- **Hand in**: Compress all your files into a single file and submit that single compressed file via the **Assignment 1 Submission** link on Canvas under "Assignments". A Readme file is not required but you may include one if there's anything you want to tell me. If you work as a group, each group needs to submit a single copy of the work. You must indicate in your submission: (1) the team members. (2) the detailed breakdown of the workload. And the division of the workload in percentage among group members.
- Read the following specs carefully!

Introduction

A binary tree provides a natural way to represent arithmetic expressions. Here we consider simple forms of expression trees formed from integers and the 5 binary operators +, -, *, / and ^ (power). The internal nodes of an expression tree contain any one of the binary arithmetic operators. And the leaves of an expression are operands. Go to lecture slides for examples of expression trees.

Input

We will input arithmetic expression as text string. In this assignment we assume the expression input to your program appear as a form of infix notation, fully parenthesized, and provided on a single line. Thus 5+2*4 is not a legal input; it must be written as (5+(2*4)). This saves you from having to deal with operator precedence rules.

Some examples of valid expressions (Yes, a single operand expression is valid!):
25, (15+8), ((7+9)*16), ((45+4)/(3*12)), ((9+(4^28))*(14+13))

A text expression is "fully parenthesized" when each arithmetic operator and its operands are surrounded by their own (unique) pair of parentheses. Spaces in these text expressions do not matter. E.g. (9   +5) is the same as (9+5)

Output (15pts)

1. Turn the input arithmetic expression into an expression tree. Output the prefix, infix, and postfix notations via preorder, inorder and postorder traversal, respectively, on the expression tree.
2. Evaluate the resulting expression tree and output the value.

Sample output for the given expression: ((3+4)*((5^6)/7)) as follows:
Output:

1. Prefix: * + 3 4 / ^ 5 6 7
   Infix: ((3+ 4)*(5^6)/7))
   Postfix: 3 4 + 5 6 ^ 7 / *
2. Value: 15625.000000000002

Note: Infix output has parentheses (don't copy over whatever your input was, instead applying the inorder traversal procedure!) but no parentheses are seen in the prefix and postfix output.

Implementations (15pts):

1. Start by writing a generic binary tree package (gen_tree.ads, gen_tree.adb) using the linked representation.
2. You should also write a TreeExpression package (tree_expression.ads, tree_expression.adb) that implements the conversion to an expression tree from a given expression, a tree expression evaluation, etc.
3. Resources you can use and modify:
   - Sample codes on http://users.cis.fiu.edu/~weiss/ada.html
   - Cite properly.
4. Comment on your submission properly. Name your files with specified names. Put all the .ads and .adb files involved in your implementation into a zip file and submit a single zip file. Feel free to reuse the stack and queue packages in your implementation.
5. Try to integrate the modularity design in your code.
6. When encountering an invalid expression such as (4 +) or ((4 +3) (2 +5)), you should throw an InvalidExpressionException.