**Lab 7 – Trees**

**CSC 3302**

|  |  |
| --- | --- |
|  |  |
| **Introduction:** | Trees are useful in the organization and manipulation of information. The tree structure may be used to model an arithmetic expression. For any given parent node, it is an algebraic operator which is applied to its children, each of which is an operand. When using trees for arithmetic expressions, preorder and postorder traversals are very useful, but inorder cannot be relied upon. Arithmetic expression trees can easily be recreated from the postorder and preorder expressions because they are created a certain way … operator followed by (or preceded by) two operands. |
|  |  |
| **Program:** | Write a program called **p7.java** that will accept a series of simple postfix expressions. Each infix expression consists of variable names/values; the arithmetic operators +, -, \*, /, and ^; and parentheses.  (a) build a binary expression tree from the postfix expression,  (b) evaluate the postfix expression,  (c) traverse the tree in preorder, printing the prefix expression.  For each infix expression, print its equivalent postfix expression, the evaluated results, and the equivalent prefix expression.  To make things relatively easy, you may assume that all infix expressions ARE valid. |
|  |  |
| **Input:** | The postfix expressions in the assignment statements may be a string of arbitrary length, terminated by an end of line character. Valid operands will be one digit (whole numbers); valid operators will be from {+, -, \*, /, ^}; The expression will NOT contain blanks (the blanks in the sample below are for readability only).  A sample data set might be:  2 3 \* 4 0 \* +  Use **p7.dat** for the postfix expressions. |

|  |  |
| --- | --- |
|  |  |
| **Output:** | After each postfix assignment expression is read (and printed), print the evaluation of the expression, and the prefix equivalent of the string with appropriate labels. |
|  |  |
| **Example:** | Given the above sample input, the following would occur:  Postfix: 2 3 \* 4 0 \* +  Result: 6  Prefix: + \* 4 0 \* 2 3 |
|  |  |
| **Hints:** | The **main** program should control the processing. Use any stack implementation as needed. Use a reference-based implementation of the ADT Binary Tree. Your stack class and tree class should be separate files. |
|  |  |
| **Data:** | On the server, you will need to use the following absolute path and append the filename to it. "/home/courses/csci3302-002/datafiles/" |
|  |  |
|  | Remember that you must pass the data file name in as a command line argument. |