**CIS 360 Lab #4: Expression Tree and Tree Traversal**

1. Implement a binary tree node class similar to the following:

*Class BinaryNode{*

*DataType key;*

*BinaryNode left;*

*BinaryNode right;*

*}*

1. Implement a program that reads in a **postfix** expression (*every operator is preceded by* ***exactly*** *two operands*), using integers as operands and basic mathematics operators (+, -, \* and /), and constructs an expression tree out of binary nodes for this expression.

The program should utilize a stack to build the tree, using the following logic:

* + *Read one symbol at a time from an input expression*
  + *If the symbol is an operand, create a one-node tree and push it to stack*
  + *If the symbol is an operator*
    - *Create a new node for the symbol*
    - *Pop two trees T1 (popped first) and T2 from the stack*
    - *Form a new tree: root(symbol), left (T2), right (T1)*
    - *Push the new tree to stack*

1. Implement the three traversal methods: pre-order traversal, in-order traversal and post-order traversal. When “visiting” a node, each method should print the node’s value.

*Void* ***pre-order-traverse*** *(node v) Void* ***in-order-traverse*** *(node v)*

*{ {*

*visit (work at) v; if(v.left != null)*

*for( each child u of v) in-order-traverse(v.left)*

*pre-order-traverse(u); visit (work at) v;*

*} if(v.right != null)*

*in-order-traverse(v.right)*

*Void* ***post-order-traverse*** *(node v) }*

*{*

*for( each child u of v)*

*post-order-traverse(u);*

*visit (work at) v;*

*}*

1. Test all of your code using two postfix expressions, each with at least 12 symbols
   1. First use your code from Task 2 to read the postfix expression and return the tree
   2. Then run each of these three traversal methods on the tree, to print out the prefix, infix and postfix expression correspondingly.