Ryan Ellis
COSC 320 --- 001 Lab 4 2/28/2025

Lab report: In this lab, we were given the tnode class structure, along with the infix2Postfix class structure and the buildExpTree function, in order to build an expression tree that would display an expression that is input by the user in prefix and postfix form, along with a printed expression tree (infix). This lab took me about 2 hours to complete cumulatively.

Pre-lab:

1. Review what we learned binary tree traversal and understand different orders of nodes: in-order, pre-order, post-order, level-order.
2. Read and understand d_tnode.h
3. Read d_tnode1.h and understand functions: inOrderOutput, postorderOutput, levelorderOutput

Lab 3.1

Exercise 1:
For the following infix expression, build the corresponding expression tree in paper (not in program yet).

1.1 a*b

1.2 a+b*c

1.3 a+b*c/d-e

Lab 4

#cosc320

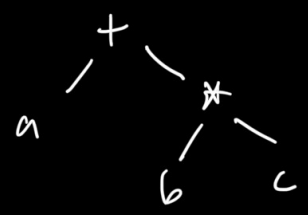
2/25

1.1 a * b ⇒

PRE (NLR): *ab

POST(LRN): ab*

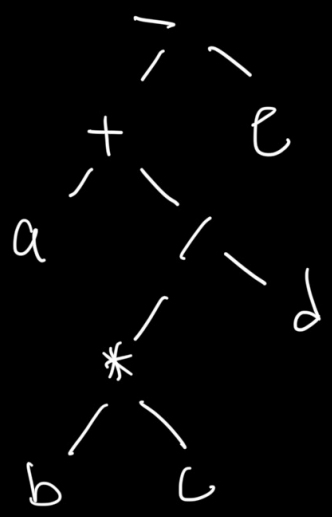
1.2 a + b * c



PRE (NLR) : +a*bc

POST(LRN): abc*+

1.3 a + (b * c / d) - e



PRE(NLR): - +a/*bcde

POST(LRN): abc*d/+e-

Perform pre-order and post-order traversal of the above binary expression trees. What relationship exists among these scans and prefix and postfix notation for the expression?

Observing the drawn trees, there is a distinct pattern between both prefix and postfix notation, where in prefix notation the operands come before the literals in the node value, and in postfix notation the operands come after the literals in the node value.

Exercise 2:

Implementation File (main program)

```
//===============================================================
// Filename: lab_04.cpp
// Author: Ryan Ellis
// Creation Date: 2/25/2025
// Last Update: 2/28/2025
// Description: Main function to build an expression tree and output the give expression the user inputs in post and pre-fix form along
// with displaying the expression tree.
// Notes:
//===============================================================
#include <iostream>
#include <stack>
#include <string>
#include "d_tnode.h"
#include "d_tnode1.h"
#include "inf2pstf.h"

using namespace std;

tnode<char> *buildExpTree(const string &);       //prototypes
void prefixOutput(tnode<char> *);
void div();

int main (){

   string expression;      //user string
   tnode<char> *exp;       // tnode for prefix, buildtree

   cout<<"Enter the expression: ";
   cin>> expression;

   infix2Postfix obj;              //infix to post fix object for post fix
   obj.setInfixExp(expression);

   exp = buildExpTree(obj.postfix());         //build tree expression in tnode object
```

```
   cout<<"Prefix form: ";                    //call to print prefix form
        prefixOutput(exp);

        div();

   cout<<"Postfix form: ";
        postorderOutput(exp, " ");           //call to print postfix form

        div();
        cout<<"Expression Tree";             //print expression tree
        div();

   displayTree(exp, 2);


   return 0;
}
void div(){
   cout<<"\n============================"<<endl;              //div line function
}

void prefixOutput(tnode <char> *exp){              //function to print tnode tree in prefix form
   if(exp){
                cout<<exp->nodeValue<< " ";
                prefixOutput(exp->left);
                prefixOutput(exp->right);
        }


}

// build an expression tree from a postfix expression.
// the operands are single letter identifiers in the range from
// 'a' .. 'z' and the operators are selected from the characters
// '+', '-', '*' and '/'
tnode<char> *buildExpTree(const string& exp)
{
        // newnode is the address of the root of subtrees we build
        tnode<char> *newnode, *lptr, *rptr;
        char token;
        // subtrees go into and off the stack
        stack<tnode<char> *> s;
        int i = 0;


        // loop until i reaches the end of the string
        while(i != exp.length())
        {
                // skip blanks and tabs in the expression
```

```
            while (exp[i] == ' ' || exp[i] == '\t')
                i++;

            // if the expression has trailing whitespace, we could
            // be at the end of the string
            if (i == exp.length())
                break;

            // extract the current token and increment i
            token = exp[i];
            i++;

            // see if the token is an operator or an operand
            if (token == '+' || token == '-' || token == '*' || token == '/')
            {
                // current token is an operator. pop two subtrees off
                // the stack.
                rptr = s.top();
                s.pop();
                lptr = s.top();
                s.pop();

                // create a new subtree with token as root and subtees
                // lptr and rptr and push it onto the stack
                newnode = new tnode<char>(token,lptr,rptr);
                s.push(newnode);
            }
            else // must be an operand
            {
                // create a leaf node and push it onto the stack
                newnode = new tnode<char>(token);
                s.push(newnode);
            }
        }

        // if the expression was not empty, the root of the expression
        // tree is on the top of the stack
        if (!s.empty())
            return s.top();
        else
            return NULL;
}
```

Output:

```
ryan@ryan-MacBookPro:~/Documents/COSC 320/Labs/Lab 4$ ./prog
Enter the expression: (a+b)/c
Prefix form: / + a b c
============================
Postfix form: a b + c /
============================
Expression Tree
============================
          /
    +       c
 a    b
ryan@ryan-MacBookPro:~/Documents/COSC 320/Labs/Lab 4$
```

```
ryan@ryan-MacBookPro:~/Documents/COSC 320/Labs/Lab 4$ ./prog
Enter the expression: a*b
Prefix form: * a b
============================
Postfix form: a b *
============================
Expression Tree
============================
    *
 a    b
ryan@ryan-MacBookPro:~/Documents/COSC 320/Labs/Lab 4$
```

```
ryan@ryan-MacBookPro:~/Documents/COSC 320/Labs/Lab 4$ ./prog
Enter the expression: a+b*c
Prefix form: + a * b c
============================
Postfix form: a b c * +
============================
Expression Tree
============================
    +
 a      *
      b    c
ryan@ryan-MacBookPro:~/Documents/COSC 320/Labs/Lab 4$
```

```
ryan@ryan-MacBookPro:~/Documents/COSC 320/Labs/Lab 4$ ./prog
Enter the expression: a+b*c/d-e
Prefix form: - + a / * b c d e
============================
Postfix form: a b c * d / + e -
============================
Expression Tree
============================
                      -
    +                   e
 a            /
         *       d
       b    c
ryan@ryan-MacBookPro:~/Documents/COSC 320/Labs/Lab 4$
```