Programming Assignment #2 (Due 11/16 23:59)

- For this programming assignment, you are required to implement an ***arithmetic expression evaluator*** using a ***stack*** that stores data in a ***doubly linked*** list.
- (40 points) First of all, you have to implement your own ***stack template class*** using a doubly linked list. You can reuse any code for the node class and list functions you've implemented for the programming assignment #1, but they have to be reimplemented as ***class/function templates*** now.
- (45 points) Then, define and implement a class named ***evaluator*** that provides three public functions:

  1. string read_expression( ) function that reads in the expression from the standard input and returns the string representing the user-provided expression.

  // You may assume that the user input is a well-formed infix arithmetic expression (i.e., no error checking required) where the following conditions are met:

  (1) all operators(+,-,*,and /) are binary,

  (2) operands are ***numbers*** or ***variables,*** and

  (3) parentheses are allowed in the expression, but not every operation needs to be fully parenthesized

  // postcondition: a string for infix expression has been returned (Example: "(1.03 * x + .4) / y")

Programming Assignment #2 - continued

2. string convert_to_postfix(string s) function that converts s (an infix arithmetic expression) into a postfix notation and returns the result string.

// precondition: s is a well-formed infix arithmetic expression (i.e., no error checking required) where the following conditions are met:

(1) all operators(+,-,*,and /) are binary,

(2) operands are ***numbers*** or ***variables,*** and

(3) parentheses are allowed in the expression, but not every operation needs to be fully parenthesized

// postcondition: a string for postfix expression converted from s has been returned and the postfix string should have at least one blank space between operands/operators. (For example, 10 2 * .4 +)

3. double evaluate_postfix(string s) function that evaluates string s which is a postfix arithmetic expression and returns the result. If a variable in the expression has to be evaluated and no value is known for it yet, then the evaluation routine should ask the user to enter its value from the standard input.

// precondition: string s is a well-formed postfix arithmetic expression (i.e., no error checking required) in which variables are allowed

// postcondition: the evaluation result for s has been returned

- (15points) Your main program needs to interact with the user to read in and evaluate infix expressions using those three functions defined above and should show the intermediate results by writing the ***input expression (in infix notation), its postfix expression, and the evaluation result*** to the standard output. And your main program should be able to repeat the same process until the user wants to quit.