CS 121 | **Programming Assignment #2**
Bruce Bolden | **20 Points**
February 22, 2023 | **Due:** March 3, 2023

Objective: Become familiar with the typical stack ADT and increase your experience with classes.

Program Description: Many programming languages use infix notation for arithmetic expressions, e.g., $3 + 4$. Since it is much easier to evaluate postfix expression (e.g., $3\,4\,+$), infix expressions are often converted to postfix expressions internally by compilers or interepreters.

Infix to postfix expression algorithm:

```
 1  Push a '(' onto the stack
 2  Add a ')' to the end of the infix expression

 3  while the stack is not empty
 4     Get (read) the next token from the infix expression

 5     if the token is a '('
 6        Push the token onto the stack
 7     else if the token is a number
 8        Add the number to the end of the postfix expression
 9     else if the token is a ')'
10        Pop the element c from the stack
11        while c is not a '('
12           Place c at the end of the postfix expression
13           Pop another element c from the stack
14     else (the token must be an operator)
15        while the top of the stack is an operator with
            precedence greater than or equal to the token
16           Pop the element c from the stack
17           Place c at the end of the postfix expression
18        Push the token onto the stack
```

Note that this algorithm assumes that the infix expression is written correctly! Getting (reading) the next token at the top of the loop the token is either a number, an operator, or a `'('` or a `')'`.

Once the infix expression has been converted to a postfix expression, the result can be evaluated using a postfix evaluation algorithm.

Before you start coding, work through several examples by hand.

All appropriate error checking should be performed including invalid operations. The program should loop continuously for input until terminated with the quit keyword. The user interface should model the application demonstrated above *exactly*. Particularly note that input is processed a line at a time.

```
$ ./in2post
Infix:   3 + 4
Postfix: 3 4 +
$ ./in2post
Infix:   (3 + 4)
Postfix: 3 4 +
```

Requirements: In addition to the functional description above, your solution also:

- **Must** have the stack implemented as a linked list class. You may use (extend) any code covered in the course.
- **Must** have the logic/implementation in a separate file from the stack implementation (an application file).
- **Must** organize the source code files in accordance with the class standards. In other words, separate class declaration files (ADT header file), class implementation files (ADT implementation file), and application file.

Deliverables:

- Program—fully documented and functional program. No wrap-around of lines on the printed copy!
- A program design. Describe all classes and methods needed to implement your program.
- Programming Log:
  - Record the time required to design and implement your program.
  - Record of things you encountered/learned while implementing your program.
- Output—proof that your program worked. Turn in the output from your program illustrating *ALL* of the basic functions described above *PLUS* the following infix expressions:

  ◦ (2 + 3) * 4
  ◦ 2 + ((5) * 3)
  ◦ 2 * ((3 + 4 * 7) / 2 + 4)

If you have any questions regarding this assignment, do not hesitate to contact me. Start working on this assignment as soon as possible.