CSCE3110 Data Structures and Algorithms
Assignment 2

Due: 06/20/2023 on Canvas
(100 points + 10 bonus points)

Instructions: Submit all files that are needed to compile and run your code on Canvas in a zip file. Include a brief README file explaining your code, especially if you implemented some of the suggestions for extra credit.

We are providing some sample code and input files:

```
public/
  balancing.cpp
    - main method to check whether an expression is balanced
  infix2postfix.cpp
    - main method to transform an infix expression into postfix
  input_balanced.txt
    - test cases for balancing.cpp
  input_infix2postfix.txt
    - test cases for infixtopostfix.cpp
  input_postfixEval.txt
    - test cases for postfixEval.cpp
  postfixEval.cpp
    - main method to evaluate postfix expressions
  stack.cpp
    - stack implementation
  stack.hpp
    - stack header file
  extra_credit
    - work on the extra credit here
```

Please use CELL machine to compile and execute your code. You can find how to use CELL machine on Canvas.

- To compile, run

  ```
  $ g++ stack.cpp balancing.cpp
  $ g++ stack.cpp postfixEval.cpp
  $ g++ stack.cpp infixtopostfix.cpp
  ```

- To run each program, run

  ```
  $ ./a.out
  ```

- The test cases follow this format: `expected_solution input`. Given `input`, your job is to implement code that gets the `expected_solution`. Obviously, you need to calculate `expected_solution`, not just print it.
- `balancing.cpp` must balance round parentheses, and square and curly brackets (() [] {})
- While we provide a few test cases, you are expected to add more to make sure your code works.

Question:

(a). (30points) Based on the stack.hpp file, implement a stack using array in the stack.cpp file.

Specifically, you need to complete the push and pop method.

Use the implemented stack to solve the following problems:

(b). (20points) Balancing parenthesis:

Complete the balancing.cpp file based on the comments in the file. Specifically, complete the for loop as shown below to check whether s is balanced:

```
23      for(int i=0; i<s.length(); ++i){
24        // WRITE CODE HERE so that isBalanced indicates whether 's' is balanced
25      }
```

(c). (20points) Evaluating postfix expressions:

Complete the postfixEval.cpp file based on the comments in the file. Specifically, complete the for loop as shown below to evaluate the postfix expression in s:

```
17      for(int i=0; i<s.length(); ++i){
18        // WRITE CODE HERE to evaluate the postfix expression in s
19        // At the end of the loop, stack.pop() should contain the value of the postfix expression
20      }
```

(d). (30points) Transforming infix expressions into postfix expressions:

Complete the infixtopostfix.cpp file based on the comments in the file. Specifically, complete the for loop as shown below to store in ‘result’ the postfix transformation of ‘input’:

```
39      for(int i=0; i<input.length(); ++i){
40        // WRITE CODE HERE to store in 'result' the postfix transformation of 'input'
41      }
42
43      // You need to do some extra stuff here to store in 'result' the postfix transformation of 'input'
```

In addition, right after the for loop, do extra stuff to store in ‘result’ the postfix transformation of ‘input’(pop all symbols from stack and insert into ooutput):

```
42
43        // You need to do some extra stuff here to store in 'result' the
    postfix transformation of 'input'
```

Extra credit (10 points): please program in the extra_credit folder.

In the extra_credit folder,

- Implement a stack using linked-list (you will need to complete the stack.cpp files).

[hint: after you implement stack using linked-list, and compile and execute as below:

- To compile, run

  $ g++ stack.cpp balancing.cpp

  $ g++ stack.cpp postfixEval.cpp

  $ g++ stack.cpp infixtopostfix.cpp

- To run each program, run

  $ ./a.out

You should see the same output as you implement the stack using array.]