Data Structures (Spring 2020)
Lab Exercise 6

Task-1: Create a C++ Program which converts an Infix expression into a Postfix expression.

Algorithm to convert Infix to Postfix[1]
Let, ***INFIX*** is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression ***POSTFIX***.

1. Push "(" onto Stack, and add ")" to the end of ***INFIX.***
2. Scan ***INFIX*** from left to right and repeat Step 3 to 6 for each element of ***INFIX*** until the Stack is empty.
3. If an operand is encountered, add it to ***POSTFIX.***
4. If a left parenthesis is encountered, push it onto Stack.
5. If an operator is encountered, then:
    a. Repeatedly pop from Stack and add to ***POSTFIX*** each operator (from top of the Stack) which has the same precedence as or higher precedence than operator.
    b. Add operator to Stack.
    [End of If]
6. If a right parenthesis is encountered, then:
    a. Repeatedly pop from Stack and add to ***POSTFIX*** each operator (from top of the Stack) until a left parenthesis is encountered.
    b. Remove the left Parenthesis from Stack.
    [End of If]
7. END.

Check your code with following infix expressions and expected outputs:

```
Enter an infix expression: ((a+b)/c)
The postfix form is: ab+c/

Enter an infix expression: 2+3*4
The postfix form is: 234*+

Enter an infix expression: 2*(A-B)+3+C
The postfix form is: 2AB-*3+C+

Enter an infix expression: 3*(4-2^5)+6
The postfix form is: 3425^-*6+
```

1 Ref: https://www.includehelp.com/c/infix-to-postfix-conversion-using-stack-with-c-program.aspx

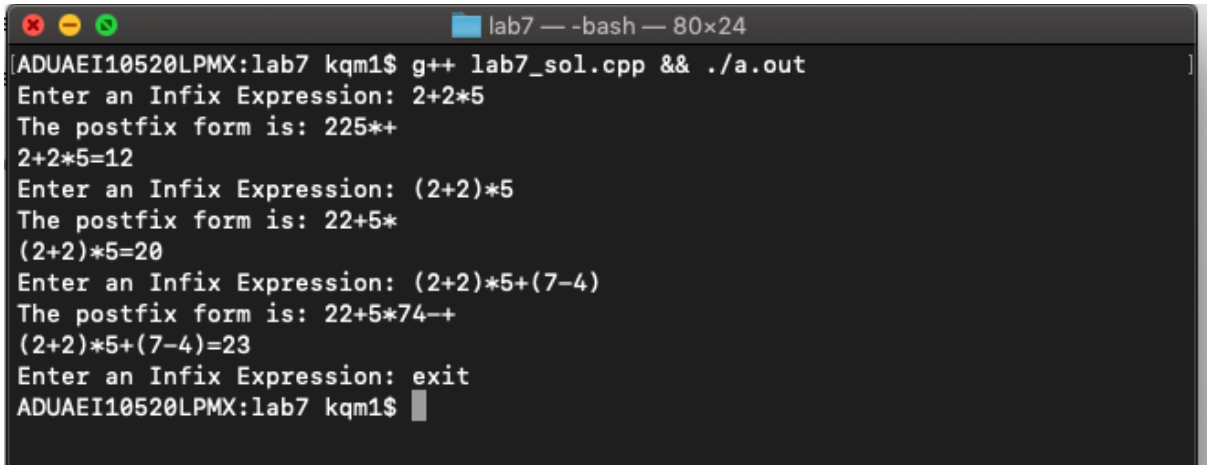
Task-2: Add a function called "**evaluate()**" to your program, which evaluate the postfix expression using a Stack and returns the result to the calling program.

Algorithm to evaluate a postfix expression

1. Create a stack to store the operands (values)
2. Scan the ***POSTFIX*** expression from left to right for every element
    a. if the element is an operand push it to the stack
    b. if the element is an operator pop 2 elements from the stack, apply the operator on it and push the result back to the stack
3. The element/value left on the stack is the final answer of the expression.

Note: For simplicity you can assume that every number and operator is only one letter long.

Output of your program should look similar to the following:



Task-3: Instead of using the STL stack, create your own stack class, called CStack, and use it for tasks 1 and 2 above. The class CStack should contains at least following Methods:

- **push(e):** Insert element e at the top of the stack
- **pop():** Remove the top element from the stack; error if empty
- **top():** Return a reference to (value of) the top element on the stack, without removing it; error if empty
- **size():** Return the number of elements in the stack.
- **empty():** Return true if the stack is empty and false otherwise.