Assignment 4

CS-3240, Section 601, Spring 2021

Purpose

Implement and use a Stack ADT to convert infix mathematical expressions to postfix, and then evaluate the postfix expressions. Input will be from a text file, and output will be written to a file.

Task

Stack ADT (stack.py)

You will implement a Stack ADT (class Stack) that supports the following operations:

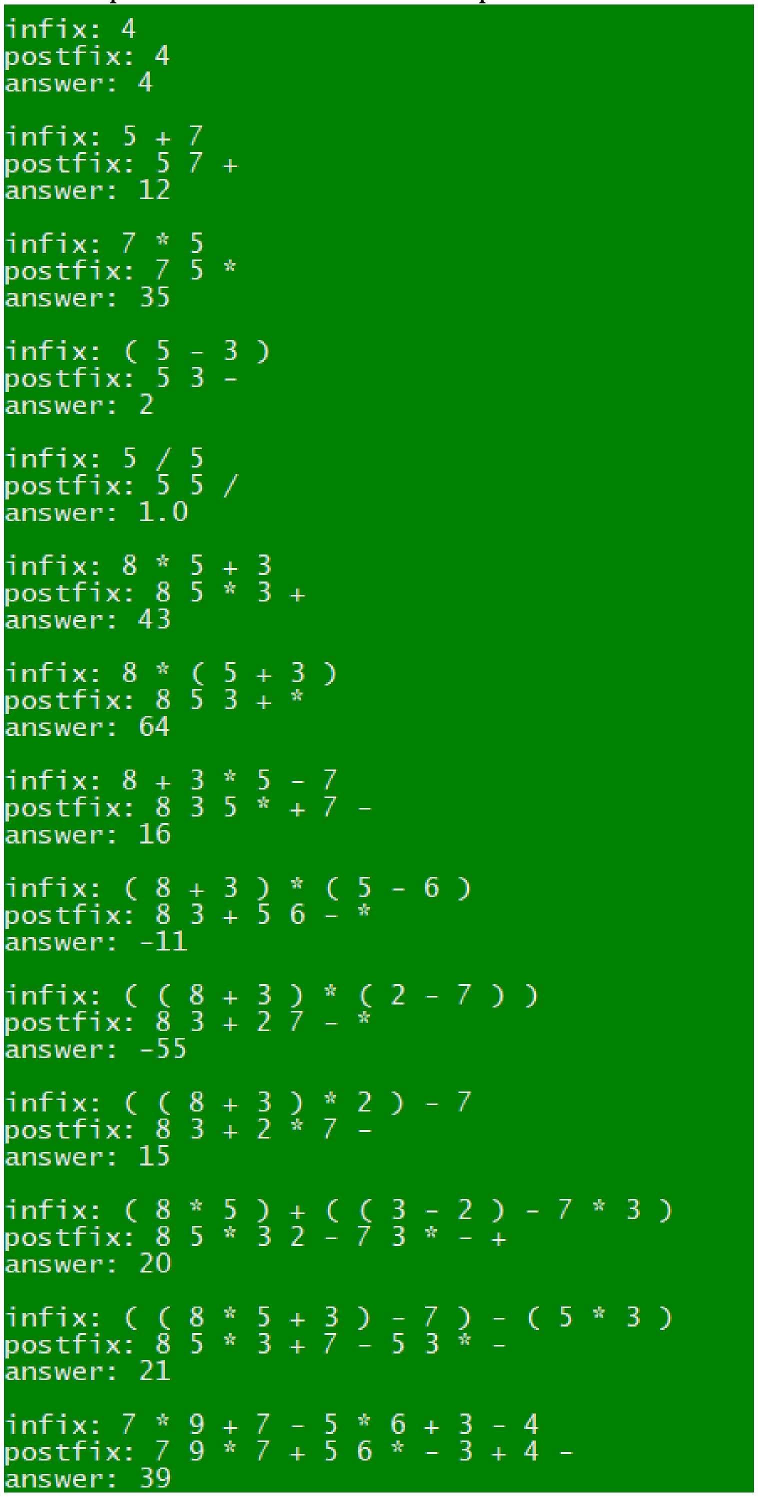
- **push(item):** push an item onto the stack. Size increases by 1.
- **pop()**: remove the top item from the stack and return it. Raise an IndexError if the stack is empty.
- **top()**: return the item on top of the stack without removing it. Raise an IndexError if the stack is empty.   We called this Peek in our in-class example.
- **size()**: return the number of items on the stack.
- **clear()**: empty the stack.

Main Program (main.py)

You will implement a main program with the following methods:

- **in2post(expr)**: takes an infix expression as an input and returns an equivalent postfix expression as a string.
  - If the expression is not valid, raise a **SyntaxError**. You can detect that an expression is not valid if your stack is empty when you are about to pop off the operator stack.
  - If the parameter expr is not a string, raise a **ValueError**. You can use *isinstance* to determine if the parameter expr is a string.
- **eval_postfix(expr)**: takes a postfix string as input and returns a number.
  - If the expression is not valid, raise a **SyntaxError**. You can detect that an expression is not valid if your stack is empty when you are about to pop a value off it.
- main(): reads in the data.text file.
  - You can use file1 = open('<filename>', 'r') to open the file.
  - You can read all lines into a list bye using file1.readlines()
  - Reads each line as an infix expression and prints that expression.

- Converts the line to a post fix expression using the *in2post* function, then prints out the postfix expression.
- Evaluates the expression using the *eval_postfix* function.
- The output should match the example below:



```
infix: 4
postfix: 4
answer: 4

infix: 5 + 7
postfix: 5 7 +
answer: 12

infix: 7 * 5
postfix: 7 5 *
answer: 35

infix: ( 5 - 3 )
postfix: 5 3 -
answer: 2

infix: 5 / 5
postfix: 5 5 /
answer: 1.0

infix: 8 * 5 + 3
postfix: 8 5 * 3 +
answer: 43

infix: 8 * ( 5 + 3 )
postfix: 8 5 3 + *
answer: 64

infix: 8 + 3 * 5 - 7
postfix: 8 3 5 * + 7 -
answer: 16

infix: ( 8 + 3 ) * ( 5 - 6 )
postfix: 8 3 + 5 6 - *
answer: -11

infix: ( ( 8 + 3 ) * ( 2 - 7 ) )
postfix: 8 3 + 2 7 - *
answer: -55

infix: ( ( 8 + 3 ) * 2 ) - 7
postfix: 8 3 + 2 * 7 -
answer: 15

infix: ( 8 * 5 ) + ( ( 3 - 2 ) - 7 * 3 )
postfix: 8 5 * 3 2 - 7 3 * - +
answer: 20

infix: ( ( 8 * 5 + 3 ) - 7 ) - ( 5 * 3 )
postfix: 8 5 * 3 + 7 - 5 3 * -
answer: 21

infix: 7 * 9 + 7 - 5 * 6 + 3 - 4
postfix: 7 9 * 7 + 5 6 * - 3 + 4 -
answer: 39
```

Criteria

The scores are a result of the Unit Tests

• Test Stack	20 pts
• Test in2post()	30 pts
• Test eval_postfix()	30 pts
• Test Code Quality	20 pts

Submission

Turn in your **stack.py** and **main.py** into canvas. You do not need to record your video for this assignment.

Additional Information

A great explanation video for these following two algorithms is found at:

https://www.youtube.com/watch?v=HJOnJU77EUs

This has a C++ flavor, but can be very helpful to Python students as well.