Infix expression: An infix expression is what we are used to seeing in math. Here we have two operands and in between the operation that we want to use on those two operands.

Example: a+b.

Postfix: Is when we write the two operands and then the operation we want to do on them. For example: ab+. It's actually more efficient for a computer to evaluate expressions in postfix, but we have trouble reading them. It's also easier to read precedence as the operator always applies to the previous two numbers.

For the sake of simplicity with this assignment, we will ignore precedence by sticking with addition and subtraction, but you are welcome to dive into it in an extension. It adds interesting complexity to the challenge.

A common use of a stack is to convert or evaluate such expressions. I would like you to use the stack structure presented in class to create three functions:

postfix_to_infix(pFix)
infix_to_postfix(iFix)
postfix_compute(pFix)

Remember you can limit yourself to +, -
You can also assume no numbers more than 1 digit

I've created a starter file for you that might provide some hints:

https://onlinegdb.com/b-tXz2SPD

Extensions:

1. Add in error handling that deals with empty stacks and invalid expressions
2. Add a infix_compute(infix)
3. Encapsulate your functions in a class objects
4. Add in the ability to handle parenthesis (you'll have to do some research on this one)
5. Add in other functionality not discussed here

Note: This is a classic problem so I suspect there are published algorithms out there. I'll be watching to make sure you don't use one of them. To prevent suspecian, make sure your code is well documented.

Report:

- What was the hardest part of this assignment?
- What was the easiest part of this assignment?
- What did you learn?
- What extensions did you do?

Rubric

infix_to_postfix tests pass: 6 points
postfix_to_infix tests pass: 6 points
post_fix_compute tests pass: 6 points
Report : 4 points
Code Quality: 4 points
Extensions: 4 points