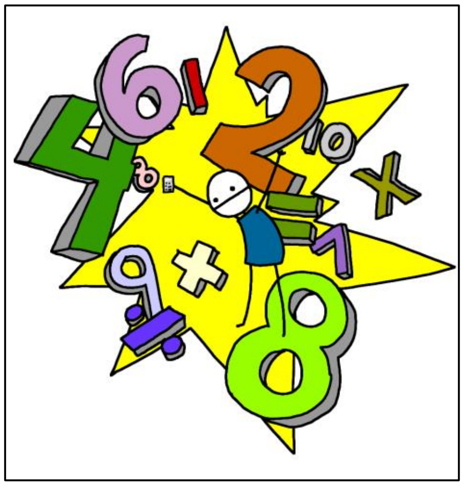
Postfix Notation



Picture from http://www.explodingdog.com/aug3/mathisfun.html

Objectives:

- To practice working with stacks
- To practice working with existing classes

Overview:

People typically use infix notation when writing mathematical expressions such as 3 + 4. A problem arises when there are several operation chained together such as 3 + 4 * 5. Should you perform the addition or the multiplication first? It makes a difference -- the answer would be 35 if the addition was first and 23 if the multiplication was first. Several hundred years ago "we" agreed on the standard "order of operations" to avoid this problem. But now we get weird expressions like (3 + 4) * 5 where we use parentheses to specify when we want to "break" the regular order of operations.

Luckily, there are other ways to write mathematical expressions. One system where there is no need for "order of operations" or parentheses is called Reverse Polish Notation, also known as postfix notation. In postfix notation, the operator always follows the operands. The following are examples of infix expressions and their equivalent postfix expressions.

Infix	**Postfix**
3 + 4	3 4 +
6 / 3	6 3 /
17 - 8	17 8 -
3 + 4 * 5	4 5 * 3 +
(3 + 4) * 5	3 4 + 5 *
(4 + 12) / 8	4 12 + 8 /

Note that there is never a need for parentheses in a postfix expression. If you want to change the order the operators are evaluated, you change the order that the operands and operators are written.

Many calculators (such as the 1970's Hewlett Packard 35) use postfix notation for input because it simplifies the calculator's work. The calculator does not have to scan an entire expression to see where the higher precedence operators are located. The calculator can perform operations as soon as they are entered.

In this lab you will write a program that evaluates an expression entered in postfix notation.

Instructions:

You will be implementing the stack class from your lecture class to assist with the evaluation of the postfix expression, if you haven't already finished it. Your Stack class will need to accommodate the following operations (from the slides):

- Initialize the Stack
- Destroy the Stack
- Is Stack Empty
- Is Stack Full
- Push
- Pop

You may assume that

- only integers are used in the expression
- only the four basic math operations (+ , -, * , /) are used in the expression
- each operation or integer will be surrounded by a least 1 space on each side

The algorithm for evaluating a postfix expression is straightforward:

```
While there are input tokens left in the expression:
    Read the next token from input.
    If the token is an operator:
        Pop the top 2 values from the stack.
        Evaluate the operator, with the values as arguments.
        Push the result onto the stack.
    Otherwise, the token is a value:
        Push the value onto the stack.
The value on the stack after processing the input is the result.
```

The program is divided in parts just as a way to help you get started. There are no actual points attached to completing parts 1 and 2.

Part 1:

Finish the Stack class. If both you and your partner have finished it, then choose which one to use.

Part 2:

Write a program so that it prompts a user to enter an expression and then evaluates that expression using the algorithm given above. You may assume that the expression entered by the user is valid to start with. However, once you have your program working with a valid expression, modify it so that it will print "Invalid expression" if an invalid expression is entered.

Part 3:

Once the program correctly evaluates postfix expressions, modify the program so that it will read expressions from a file, one expression per line, and print the answer for each to the screen. The program should prompt the user for the name of the file. The file provided is similar (but not identical) to the one used to grade your program. The actual numbers and mathematical operators will be different, but the number of values and the number of operators as well as the order of values and operators will be the same between the two files.

Program Requirements:

For the code:

- Your program should have the correct comment block at the top (see last assignment)

- Use appropriate comments throughout the code. Be sure to include comments before each function definition explaining the interface (what must be sent, what is returned) and the purpose of the function. Also, don't forget to make appropriate comments within the method definition.

- Make good use of whitespace. Be sure to include a blank line of space before each comment. Two lines of space between functions is nice, but one will suffice – whichever you choose, be consistent. Follow the Python programming style guide: https://www.python.org/dev/peps/pep-0008/#introduction

For the lab report, follow the Lab Report Format Guide and complete the following sections:

- Title Page
- Analysis and Conclusions
- Appendix B - code (Be sure that you copy and paste the code into your word document, don't take a screen shot of it and paste that – it ends up being too hard to read.)

Deliverables:

Electronic submission in myCourses:

- Code (due by the end of the class period)
- Lab Report (due by the start of the next time that your lab meets) -- either a Microsoft Word document or a pdf

Paper submission in class:

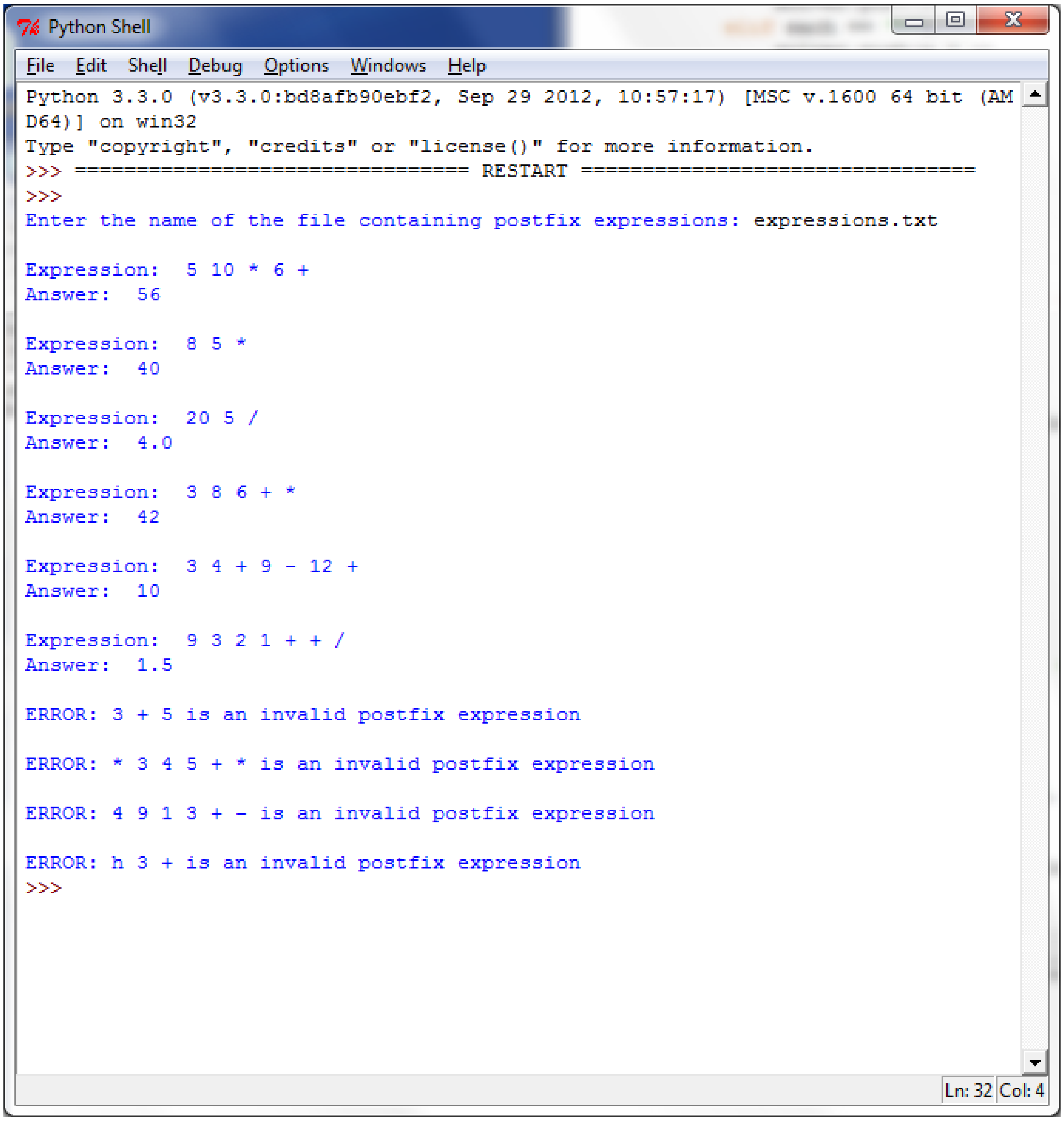
- Lab Report (due at the start of lab the next time that it meets)

Grading:

The file contains 10 expressions. Getting the right output for each expression is worth 7 points. (Total of 70 points).

Task	**Points**
Lab Report	30 points
Title Page	5 points
Analysis and Conclusions	10 points
Coding Style	15 points
Code	70 points

Sample Execution:



```
Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 29 2012, 10:57:17) [MSC v.1600 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Enter the name of the file containing postfix expressions: expressions.txt

Expression:  5 10 * 6 +
Answer:  56

Expression:  8 5 *
Answer:  40

Expression:  20 5 /
Answer:  4.0

Expression:  3 8 6 + *
Answer:  42

Expression:  3 4 + 9 - 12 +
Answer:  10

Expression:  9 3 2 1 + + /
Answer:  1.5

ERROR: 3 + 5 is an invalid postfix expression

ERROR: * 3 4 5 + * is an invalid postfix expression

ERROR: 4 9 1 3 + - is an invalid postfix expression

ERROR: h 3 + is an invalid postfix expression
>>>
```