**CS3723 Pgm 6 Python (30 pts)**

In this assignment, we will use Python to evaluate prefix expressions. Your algorithm will be recursive.

Example Expressions:

(+ (\* 2 3) (\* 4 5))

10

We must be able to recognize tokens in the expression. Notice that we can't simply split the expression on spaces since "(" and "+" are separate symbols, but are not separated by spaces.

Place all your code in **p6.py**

You must create the **prefixReader** function which does the following:

* Reads each text line in the input file (which was passed as a command argument). For each text line:
  + It should print the prefix text string, preceded by "> "
  + It **must tokenize** the expression using Python's **regex** (use the re module). The Python Part 4 notes will be useful.
  + Reset the global current parsing pos to 0
  + It should invoke **prefixEval** passing the token array. If that function returned successfully (i.e., it didn't raise an exception), it should print the value returned from prefixEval.
  + It should provide a **try… except** block which prints errors. After printing an error do not exit the program.

You must create the **prefixEval** function which is passed a token array and modifies the global current parsing pos:

* Based on the token at the current position, if if is a:

( It is evaluating a function. prefixEval should treat the next token as a function name. It now needs to evaluate the arguments for that function:

* + - * Assume the function has only two arguments. See extra credit for handling a variable number of arguments.
      * It should invoke prefixEval to get the value of each argument (appropriately advancing the global current parsing position).
      * It should advance the global current parsing position past its corresponding ")".
      * It should invoke evalOperator passing the function and arguments and return that value as prefixEval's functional value.

*number* It should return that number's integer value as prefixEval's functional value.

* prefixEval should use another function to actually apply the function to the operands. The following functions must be supported:

+ addition

* minus - subtracts the second operand from the first

\* multiplication

/ division - divides the second operand into the first, truncating the result

> greater than (numeric)

< less than (numeric)

and Boolean and returns True or False

or Boolean or returns True or False

* Before returning, prefixEval advances the global current parsing position to the position immediately after its expression

**Notes:**

1. Your program should be passed one argument - the name of the input file:

python3 **p6.py p6Input.txt** > p6Out.txt

2. Larry provided sample data in **p6Input.txt**. Extra credit data (which includes all the data in p6Input.txt) is in **p6Extra.txt**

3. Your code should **raise** several **exceptions** which should be caught by **prefixReader**. Each should be passed an appropriate error message. Example:

**raise FunctionError**("Unknown function in prefix expr: '"+ func +"'")

To print it in a try … except:

except (*Exception1, Exception2, …*) as e:

print( str(e.args[1]))

The exceptions to be raised and corresponding possible messages are shown:

**FunctionError**

* + - Unknown function in prefix expr: '*name*'
    - Incorrect number of operands - must be 2 for '*func*'
    - Incorrect number of operands - must be at least 2 for '*func*' (only for extra credit)

**PrefixSyntax**

* + - Missing closing ')'
    - Expected int found: '*str*'

4. The int(*str*) function raises a ValueError if the value isn't a valid integer. Use try …. except to catch that and raise a PrefixSyntax error passing an appropriate message.

5. Turn-in a zip file named LastnameFirstname.zip containing:

* p6.py - your source code containing all of the source
* p6Out.txt - your output

Also provide a note in BlackBoard specifying whether you did the extra credit.

**Extra Credit (3+100/N)**

1. Instead of supporting just two operands, support a variable number of operands. For example:

(+ 2 3 (\* 4 5) 6)

Note that the right parenthesis indicates the end of the arguments.

1. These operators can have a variable number of operands:

+ \* and or

1. Extra credit is **NOT** given to **late** assignments.
2. All requirements must be met to receive extra credit.
3. N is the number of people to meet all requirements on time.

Sample Partial Output:

> (\* 2 15)

30

> (- (\* 12 2)(\* 2 3 ))

18

> (\* 5 (/ 5 2))

10

> (or (> 6 13) (< 15 2))

False

> (and (> 13 6) (> 17 3))

True

> (+ 1 (\* 12 4 )(\* 2 2)

Missing closing ')'

> (/ (+ 3 5))

Incorrect number of operands - must be 2 for '/'