pl_hw3

WorkType	HW
Due Date	@March 24, 2023
Status	Review

1. (50 point) Write a lexical analyzer that recognizes all of the tokens necessary for mathematical operations:

💡 https://github.com/Palpeleno/PalProgLingistics/tree/pl_hw3

```
import re

#Recognize a single token by pointing to the current object using the symbol and name of t
oken, symbol is token type, value is the name
class Token:
    def __init__(self, token_type, value):
        self.token_type = token_type
        self.value = value

#output
    def __repr__(self):
        return f"({self.token_type}, '{self.value}')"

#file reader not working ?!something file not found when its path and name are right ther
e!!
class Lexalyzer:
    def __init__(self, file_path):
        with open(file_path, 'r') as f:
            self.text = f.read()

#tokens for math operation
    def tokenize(self):
        tokens = []
        regex_patterns = [
            (r'➕', 'ADD'),
            (r'➖', 'SUBTRACT'),
            (r'✖', 'MULTIPLY'),
            (r'➗d', 'DIVIDE'),
            (r'%', 'MODULO'),
            (r'\(', 'LEFT_PAREN'),
            (r'\)', 'RIGHT_PAREN'),
```

```
            (r'→', 'ASSIGNMENT'),
            (r'==', 'EQUALS'),
            (r'<', 'LESS_THAN'),
            (r'≤', 'LESS_THAN_EQUAL'),
            (r'>', 'GREATER_THAN'),
            (r'≥', 'GREATER_THAN_EQUAL'),
            (r'&', 'LOGICAL_AND'),
            (r'\|', 'LOGICAL_OR'),
            (r'[_a-zA-Z][_a-zA-Z0-9]*', 'IDENTIFIER'),
            (r'[-+]?[0-9]*\.?[0-9]+', 'NUMBER')
        ]

        combined_pattern = '|'.join('(?P<%s>%s)' % pair for pair in regex_patterns)
        token_regex = re.compile(combined_pattern)
        for match in token_regex.finditer(self.text):
            token_type = match.lastgroup
            token_value = match.group(token_type)
            tokens.append(Token(token_type, token_value))
        return tokens

#instance of lexalyzer class, not working, cant open file
mrlexer = Lexalyzer('test.txt')
tokens = mrlexer.tokenize()
print(tokens)
```

---

2. (5 points) Write a CFG for PEMDAS/BODMAS, make sure this CFG is not ambiguous. Make sure all binary operators are Left associative.

<expression> → <term> | <expr> <addOperator> <term>
<term> → <factor> | <term> <multiplyOperator> <factor>
<factor> → <number> | (expression) | <factor>
<number> → <digit> | <number> <digit>
<digit> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<addOperator> → ➕| ➖
<multiplyOperator> → ✖️ | ➗

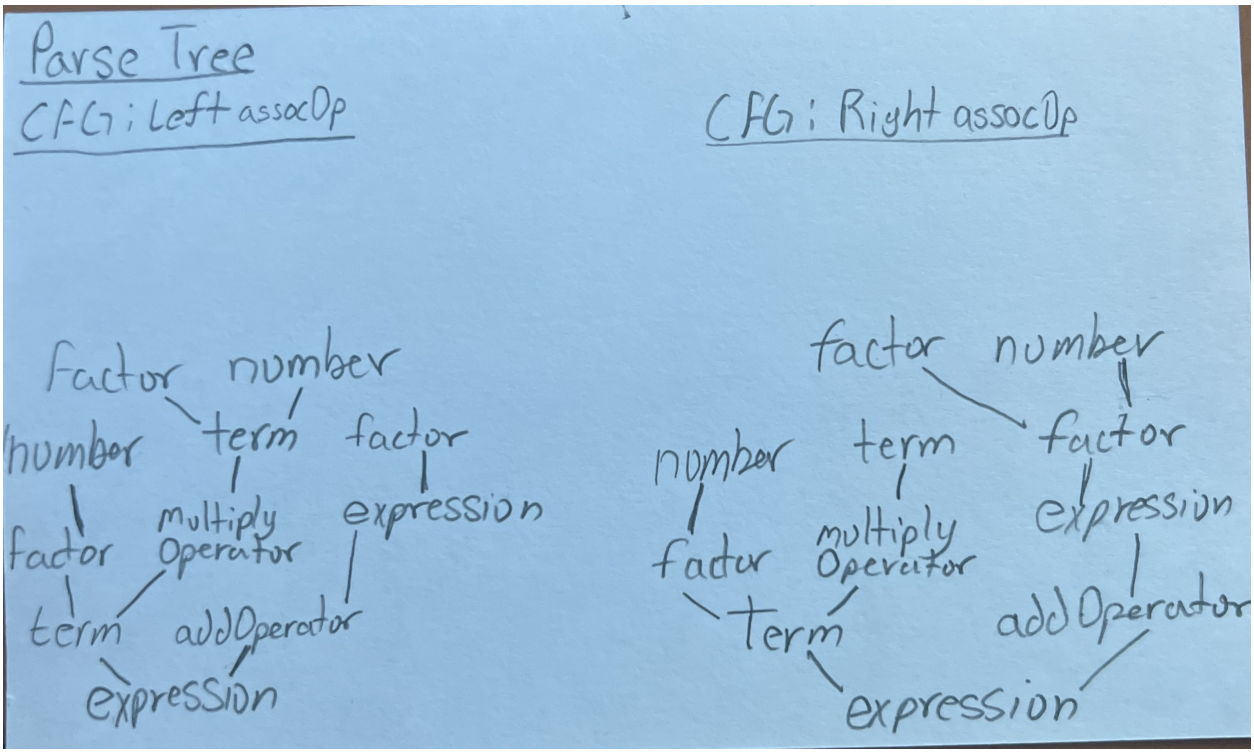
---

3. (5 points) Rewrite the above (problem 2) with only right associative operators.

<expression> → <term> | <term> <addOperator> <expression>
<term> → <factor> | <factor> <multiplyOperator> <term>
<factor> → <number> | (expression) | <factor>

<number> → <digit> | <number> <digit>
<digit> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<addOperator> → ➕| ➖
<multiplyOperator> → ✖ | ➗

4. (10 points) Draw parse trees for problems 2 and 3 using the following string:a * 12 / (x - 7.0)REMEBER EVERY VARIABLE NAME IS JUST AN IDENTIFIER



5. (10 points) Use the eBNFs rules for GO LANGS FOR LOOP to write the 10 smallest possible for loops. Treat `PrimaryExpr` as a terminal symbol.

```
#1 basic for loop with body statements
for expression; expression ; expression {
  expression
}

#2 no conditional statement
for expression;  ;  {
  expression
}

#3 no statement but body
```

```
for ; ;  {
  expression
}


#4 for loop with many inilized statments
for expression ,expression,  expression ; expression {
  expression
}

#5 more increment statements
for expression; expression ; expression, expression  {
  expression
}

#6 body with initilized statement
loop:
for expression; expression; expresion{
  expression
  if expression{
    break loop
  }
}

#7 function call as iterator statement
for expression; expression; expression(){
  expression
}

#8 loop with range for increment
for , expression = range expression {
  expression
}

#9 loop with boolean condition
for expression ; expression == expression ; expression {
  expression
}

#10 small loop no post expression
for expression ; expression; {
  expression
}
```

6. (20 point) Draw 4 parse trees for any off your answers for 5.
    a. #2 for → expression (;) → expression (;)→ expression → expression
    b. #10 for → expression (;) → expression (;) → expression

c. #3 for → (;) → (;) → expression

d. #4 for → expression → expression → expression (;) → expression (;) → expression