METROPOLITAN STATE UNIVERSITY OF DENVER
Department of Mathematical and Computer Sciences

CS 320 - Principles of Programming Languages (Fall 2019)

Homework 03

Consider the grammar below specified using EBNF notation.

```
<expression> → <expression> (+|-) <term> | <term>
<term>       → <term> (*|/) <factor> | <factor>
<factor>     → ( <expression> ) | <identifier> | <literal>
<identifier> → <letter> { (<letter> | <digit> )
<literal>    → <digit> { <digit> }
```

The table identifies the lexical unit types (tokens) of the grammar.

Token #	Description
1	Addition Operator
2	Subtraction Operator
3	Multiplication Operator
4	Division Operator
5	Identifier
6	Literal
7	Open Parenthesis
8	Close Parenthesis

Write a lexical analyzer for the language described by the grammar. You can write your solution in any language you feel comfortable with, as long as you use the interface described in [https://github.com/thyagomota/19FCS3210/tree/master/activity05]. The output of your lexical analyzer should be a list of pairs containing a lexical unit followed by its token number, in the order of their appearance. Starting on the next page are some source codes (with expected outputs) for you to try.

Zip your code and submit it through Blackboard. I will not accept submissions by email.

source1.exp

```
52 + x2 * 231 / (y3 - 8 )
```

output:

```
52 Token.LITERAL
+ Token.ADD_OP
x2 Token.IDENTIFIER
* Token.MUL_OP
231 Token.LITERAL
/ Token.DIV_OP
( Token.OPEN_PAR
y3 Token.IDENTIFIER
- Token.SUB_OP
8 Token.LITERAL
) Token.CLOSE_PAR
```

source2.exp

```
ze99 + (  3 - xyz3a      ) * 2931
```

output:

```
ze99 Token.IDENTIFIER
+ Token.ADD_OP
( Token.OPEN_PAR
3 Token.LITERAL
- Token.SUB_OP
xyz3a Token.IDENTIFIER
) Token.CLOSE_PAR
* Token.MUL_OP
2931 Token.LITERAL
```

source3.exp (two tabs after “+”, new lines, and multiple letters “abc”)

```
x +		4

- 2 / abc + 2
```

output:

```
x Token.IDENTIFIER
+ Token.ADD_OP
4 Token.LITERAL
- Token.SUB_OP
2 Token.LITERAL
/ Token.DIV_OP
```

```
abc Token.IDENTIFIER
+ Token.ADD_OP
2 Token.LITERA
```

source4.exp (with unrecognized symbols)

```
x > 4 * c
```

output:

```
Exception: Lexical Analyzer Error: unrecognized operator found
```