PLC Test 2 – Jason Hess

First of all, this is absolutely unfinished, and I am an idiot.

Token generation was easy.

Doing stuff with these tokens SHOULD have been easy, but I am an idiot.

I did not realize my mistakes until too late. Enjoy.

THE LANGUAGE:

jason (not JSON) requires spaces between every lexeme. This is to make my life easier.

A. All rules for matching lexemes with tokens are basically regular expressions (most are in the form of IF (LEXEME) == “LEXEME” such as:

```
if (string[i] == '<jason>'):
      lex_tok_list.append([string[i],"START"])
```

yar = 1 byte, shart = 2 bytes, ent = 4 bytes, lawn = 8 bytes

This creates tokens in order. Variable names use a regular expression that matches if only letters and underscores are in the name but are first checked to make sure length is greater than 5 and less than 9.

B. Production rules and explanation of other questions are below.

PSADME

```
# RULES OF PRODUCTION RULES:
    # Curly brackets '{' or '}' mean ONE OR MORE REPETITIONS
    # Square brackets '[' or ']' mean ZERO OR MORE REPETITIONS
    # '\' is an escape character, so take the next symbol literally.
    # '|' serves to mean 'OR', allowing alternative options.
    # THANK YOU   <3

# PRODUCTION RULES:
    # <program> :  JASON <statement_list> NOSAJ

    # <statement_list> :  \{ { <statement> ; } \}
    # <statement> :  <declaration> | <if_statement> | <while_statement> | <ass>

    # <declaration> :  <id> VARNAME
    # <id> :  CHAR | INT | SHORT | LONG

    # <expression> :  <term> { (MULT | DIV | MOD) <term> }
       # Mult, div, modulo, are lower precedence
    # <term> :  <factor> { (ADD | SUB) <factor> }
       # Add and subtraction are higher precedence

    # <bool_expression> :  <band> [ OR <band> ]
    # <band> :  <beq> [ AND <beq> ]
    # <beq> :  <brel> { ( DNEQ | EQ ) <brel> }
    # <brel> :  <expr>

    # <if_statement> :  EF \( <bool_expression> \)  [ LS <statement> ]

    # <while_statement> :  YL \( <bool_expression> \) <statement_list>

    # <ass> :  VARNAME = <factor>

    # <factor> :  CHAR | INT | SHORT | LONG | NUMBER | \( <expression> \)
       # Expressions in parentheses take highest precedence
```

C. All rules are from left to right and are leftmost derivations

D. There are no ambiguous rules. (No duplicate nonterminals, no possible multiple derivations)

E. This is done, I think. Instead of terminating on errors, I keep going (to report any other errors), but do not output the generated tokens.

F. This is in progress.

G. 4 test files are created and included.

testcode1.txt and testcode2.txt will pass.

testcode3.txt has 4 lexical errors, described in the file

testcode4.txt has 4 syntax errors, also described in the file.

code.txt was my original ambition, but I am not smart enough yet.

H. Unfinished. ☹