JHU Computer Organization Module 10 Exercise

Building a Scanner/Parser

In this assignment you will use PLY, a scanner and parser generator tool to create a scanner and parser for a grammar that is a slight modification to the one in last week's homework (shown below). PLY is a Python version of the popular C tools lex and yacc. Note that using a generator is a simplified way of creating a scanner or parser and is much easier than writing either one manually.

Begin by reading the following sections of the PLY documentation (found here). The docs for PLY are very well-written and easy-to-understand. I recommend reading the material closely and tracing through the examples as this will help immensely when you write your scanner/parser generator.

- Introduction
- PLY Overview
- Lex
  - Lex example-Literal Characters
- Parsing Basics
- Yacc
  - An example-Changing the starting symbol

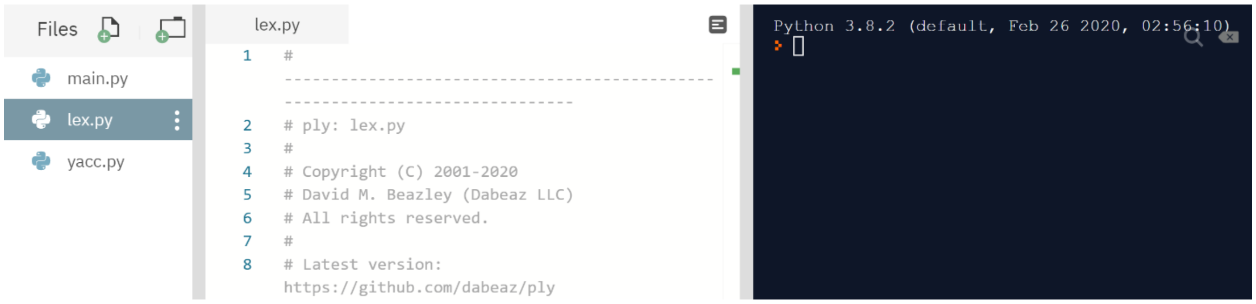
You will generate a scanner (lexer) and parser for the following grammar:

```
<stmt> : <assign> | <binop> | <declare>

<assign> : <id> '=' {<term> | <str>}

<binop> : {<term> (<literal> | <exp>) <term>} | {<str> '+' <str>}

<declare> : <var> {<id> | <assign>}

<term> : <id> | <num> | <binop>

<literal> : '+', '-', '*', '/'

<id> : ID

<num> : NUM

<exp> : EXP

<str> : STR

<var> : 'VAR '
```

```
'''binop : {term(literal | exp})term} |{str + str}'''
   '''declare: var {id|assign}'''
   '''term: id| num | binop'''
   '''literal: ‘”
```

Environment Setup

You may use any Python environment you'd like but I recommend creating a web-based REPL. You can find the source code for the PLY lexer (lex.py) and parser (yacc.py) generators here. Add two files to your REPL project named lex.py and yacc.py. Copy-and-paste the code from lex.py and yacc.py from the PLY repo into the lex.py and yacc.py files in your repo:



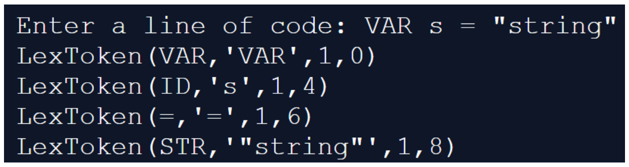
```
# -----------------------------------------------------------------------------
# ply: lex.py
#
# Copyright (C) 2001-2020
# David M. Beazley (Dabeaz LLC)
# All rights reserved.
#
# Latest version: https://github.com/dabeaz/ply
```

```
Python 3.8.2 (default, Feb 26 2020, 02:56:10)
>
```

Building a Scanner

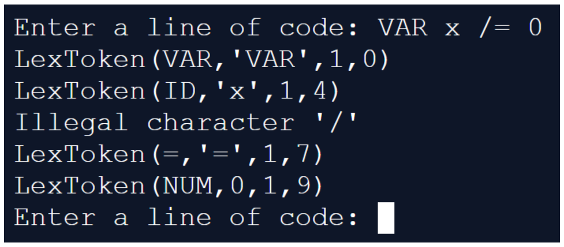
You will do your development in main.py. Copy the skeleton code for lexer_outline.py from my public repo into you're the main.py file in your REPL. You will implement the methods (I recommend one at a time) that map to each of the tokens that have been defined for you on lines 4-9. Comments have been provided for you to help guide you in creating your parser. To test, run your REPL. You will be prompted to input a line of code. Here are two sample outputs from the completed scanner:

1. A line of code containing all valid tokens:



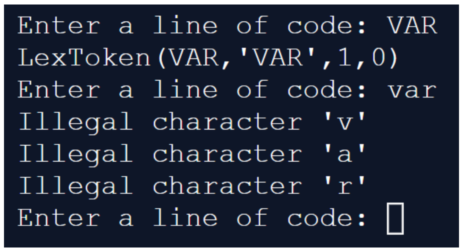
```
Enter a line of code: VAR s = "string"
LexToken(VAR,'VAR',1,0)
LexToken(ID,'s',1,4)
LexToken(=,'=',1,6)
LexToken(STR,'"string"',1,8)
```

2. A line of code containing an invalid token caught by the scanner:



```
Enter a line of code: VAR x /= 0
LexToken(VAR,'VAR',1,0)
LexToken(ID,'x',1,4)
Illegal character '/'
LexToken(=,'=',1,7)
LexToken(NUM,0,1,9)
Enter a line of code: 
```

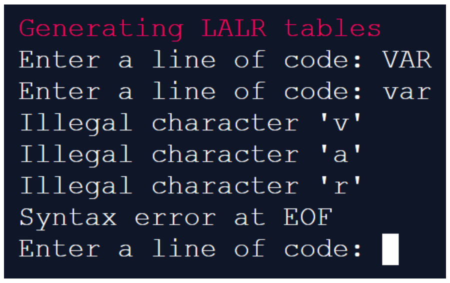
A PLY sample for generating a scanner for the 'VAR ' token has been provided for reference here. I recommend running it first and understanding how it works before adding additional tokens to it to complete your scanner. Here is an example of the code in action on both a valid and an invalid token:



```
Enter a line of code: VAR
LexToken(VAR,'VAR',1,0)
Enter a line of code: var
Illegal character 'v'
Illegal character 'a'
Illegal character 'r'
Enter a line of code: 
```

Building a Parser

Next you'll add on to your working scanner generator code to generate a full lexical and syntactic analyzer. You can run my sample scanner/parser generator using the code provided here. Here is an example of both a valid and invalid syntactical statement:



```
Generating LALR tables
Enter a line of code: VAR
Enter a line of code: var
Illegal character 'v'
Illegal character 'a'
Illegal character 'r'
Syntax error at EOF
Enter a line of code: 
```

You'll see in the code that 'stmt' is the entry point into the syntax analyzer and maps to a VAR token (which was defined in the scanner you built above). The parser that is generated checks that the input matches a VAR token and throws a syntax error if it does not.

Your job in this portion of the assignment is to combine your scanner generator code with the additional methods needed to enforce the grammar specified above (assign, declare, binop, etc.). For reference, my solution contains 5 methods to implement the grammar. Please note that the sample parser code is just an example to show you how the YACC portion of PLY works and VAR may not (probably should not) be a part of 'stmt' for your final solution.

Deliverables

1. parser_jhedid.py: a Python file containing the code for generating tokens and for performing syntax analysis against the grammar (most likely this will be the code from the main.py in your REPL)
2. A PDF named syntax_jhedid.pdf containing answers to the following question:

   "For each of the following statements, indicate whether they are VALID or INVALID for the given grammar. If you choose INVALID, please state whether the error is LEXICAL or SYNTACTIC and explain why."

   - `x = 0;`
   - `VAR x = y ** 2`
   - `VAR VAR = x/2`
   - `VAR x = VAR y = 3`
   - `x = "string" + "123"`
   - `"VAR" X = "str"`
   - `i = (2*2)**3`
   - `VAR x = 1 + 2 * 3 / 5 **6`
   - `VAR x = 1 + 2 * 3 / 5 **6 = y`
   - `x=1*2/3**4`

Rule 0	**S' -> stmt**
Rule 1	**stmt -> assign**
Rule 2	**stmt -> binop**
Rule 3	**stmt -> declare**
Rule 4	**assign -> ID = term**
Rule 5	**assign -> ID = STR**
Rule 6	**binop -> term literals term**
Rule 7	**binop -> term EXP term**
Rule 8	**binop -> STR + STR**
Rule 9	**declare -> VAR ID**
Rule 10	**declare -> VAR assign**
Rule 11	**term -> ID**
Rule 12	**term -> NUM**
Rule 13	**term -> binop**

x=0;   错误 LEXICAL error
- **VARx=y**2。正确**
- **VAR VAR = x/2 错误Syn error**
- **VARx=VARy=3 错误 syntactic error**
- **x = "string" + "123" 错误 LEXICAL err**
- **"VAR" X = "str" 错误 syntactic error**
- **i = (2*2)**3错误 LEXICAL error**
- **VARx=1+2*3/5**6正确**
- **VARx=1+2*3/5**6=y 错误 syntactic erro**

x=1*2/34正确**

```
import sys
sys.path.insert(0, "../..")

tokens = ('ID',   # ID represents a variable name
        'NUM',  # NUM represents the signed integer data type
        'EXP',  # EXP represents the exponentiation operator (**)
        'STR',  # STR represents the string data type
        'VAR', # var represents the literal string 'VAR'
        'PLUS',
        'MINUS',
        'TIMES',
        'DIVIDE',
        )

literals = ['=']

# Regular expression rules for simple tokens
t_PLUS    = r'\+'
t_MINUS   = r'-'
t_TIMES   = r'\*'
t_DIVIDE  = r'/'

# create a method for parsing VAR tokens
  # regex for finding VAR tokens
  # return token

def t_VAR(t):
  r'VAR'
```