Module 10 Assignment: Building a Scanner /Parser

EN.605.204 Computer Organization

Assignment

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

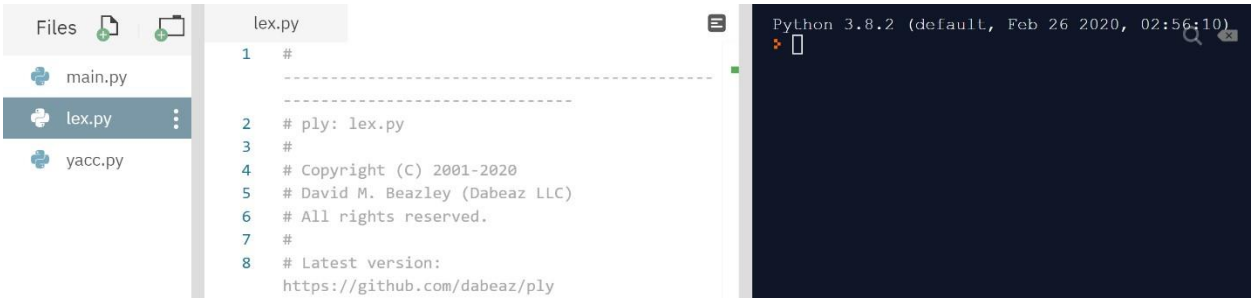
<var> : VAR

Module 10 Assignment: Building a Scanner /Parser

EN.605.204 Computer Organization

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:



Building a Scanner

You will do your development in main.py. Copy the skeleton code for **lexer_outline.py** into your 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:
```

Module 10 Assignment: Building a Scanner /Parser

EN.605.204 Computer Organization

A PLY sample for generating a scanner for the 'VAR ' token has been provided for reference **lexer_example.py** 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 the sample scanner/parser generator using **parser_example.py** 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.

Module 10 Assignment: Building a Scanner /Parser

EN.605.204 Computer Organization

Deliverables

1. **<Your JHEDID>_parser.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. Use the statements below to test your program. You should take screenshots and include them in a PDF named **<Your JHEDID>_syntax.pdf**.
3. Additionally, within the PDF named **<Your JHEDID>_syntax.pdf**, provide 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.

   1. x = 0;

      INVALID, because our grammar doesn't support a ';' at the end, this is lex error

      ```
      Enter a line of code: x = 0;
      Illegal character ';'
      Enter a line of code:
      ```

   2. VAR x = y ** 2

      INVALID, because y is not defined, syntax error because y is saved as string

      ```
      Enter a line of code: VAR x = y ** 2
      Traceback (most recent call last):
        File "main.py", line 89, in <module>
          yacc.parse(s)
        File "/home/runner/ScannerandParser/venv/lib/py
      thon3.8/site-packages/ply/yacc.py", line 333, in
      parse
          return self.parseopt_notrack(input, lexer, de
      bug, tracking, tokenfunc)
        File "/home/runner/ScannerandParser/venv/lib/py
      thon3.8/site-packages/ply/yacc.py", line 1120, in
       parseopt_notrack
          p.callable(pslice)
        File "main.py", line 66, in p_expression_binop
          p[0] = p[1] ** p[3]
      TypeError: unsupported operand type(s) for ** or
      pow(): 'str' and 'int'
      ```

3. VAR VAR = x/2

   INVALID, the second VAR here served as ID which looks really bad but is allowed. The real problem is again: x is saved as string(not defined), Syntax error

```
Enter a line of code: VAR VAR = x/2
Syntax error at 'VAR'
Traceback (most recent call last):
  File "main.py", line 89, in <module>
    yacc.parse(s)
  File "/home/runner/ScannerandParser/venv/lib/py
thon3.8/site-packages/ply/yacc.py", line 333, in
parse
    return self.parseopt_notrack(input, lexer, de
bug, tracking, tokenfunc)
  File "/home/runner/ScannerandParser/venv/lib/py
thon3.8/site-packages/ply/yacc.py", line 1120, in
 parseopt_notrack
    p.callable(pslice)
  File "main.py", line 64, in p_expression_binop
    p[0] = p[1] / p[3]
TypeError: unsupported operand type(s) for /: 'st
r' and 'int'
>
```

4. VAR x = VAR y = 3

   INVALID, syntax error, we can't assign a declaration

```
Enter a line of code: VAR x = VAR y = 3
Syntax error at 'VAR'
Enter a line of code:
```

5. x = "string" + "123"

   VALID, the "123" here is a string

```
Enter a line of code: x = "string" + "123"
Enter a line of code:
```

6. "VAR" X = "str"

   INVALID, syntax error, "VAR" here is not the legal prefix we are looking for, so it doesn't follow the instructions for declaration

```
Enter a line of code: "VAR" x = "str"
Syntax error at 'x'
Enter a line of code:
```

7. i = (2*2)**3

   INVALID, syntax error, our grammar doesn't support ()

```
Enter a line of code: i = (2*2)**3
Illegal character '('
Illegal character ')'
Enter a line of code:
```

8. VAR x = 1 + 2 * 3 / 5 **6

   VALID, our grammar support recursively binop since term can be binop

```
Enter a line of code: VAR x = 1 + 2 * 3 /5 ** 6
Enter a line of code:
```

9. VAR x = 1 + 2 * 3 / 5 **6 = y

   INVALID, syntax error, our grammar doesn't support "=y" at the end, also, it makes no sense even in human language

```
  c = re.compile('(?P<%s>%s)' % (fname, _get_regex(f)), self.reflags)
Enter a line of code: VAR x = 1 + 2 *3 /5 **6 = ySyntax error at '='
Enter a line of code:
```

   .

10. x=1*2/3**4

    VALID, this is a legal statement since statement can be assignment and then, tem can be binop.

```
Enter a line of code: x = 1 *2 /3 **4
Enter a line of code:
```

4. All documents must be submitted at the same time. You can select multiple documents by holding the <ctrl> key and clicking on the files to be submitted.