CSE 425: Programming Language Concepts

ASSIGNMENT

August 27, 2018

Farhana Haque
ID: 151 1395 042
Section: 04
Summer'18
North South University

Lexer File (bottlecap.py)

```
import ply.lex as lex
import ply.yacc as yacc
import sys

rkw = {
    'Print' : 'PRINT',
    'Add' : 'ADD',
    'with' : 'W',
    'Subtract' : 'SUB',
    'from' : 'FROM',
    'Multiply' : 'MUL',
    'Divide' : 'DIV',
    'by' : 'BY',
    'Assign' : 'ASSIGN',
    'to' : 'TO'

}
tokens = [
    'ID',
    'NUMBER',
    'PLUS',
    'MINUS',
    'TIMES',
    'DIVIDE',
    'OBRACE',
    'CBRACE',
    'NEXT'
 ] + list(rkw.values())

literals = ['\t', '\n']

t_ignore = r' '

t_PLUS = r'\+'
t_MINUS = r'\-'
t_TIMES = r'\*'
t_DIVIDE = r'\/'
t_OBRACE  = r'\{'
t_CBRACE  = r'\}'
t_NEXT = r'\t+'
```

```
def t_NUMBER(t):
    r'\d+'
    t.value = int(t.value)
    return t


def t_ID(t):
    r'[a-zA-Z_][a-zA-Z0-9_]*'
    t.type = rkw.get(t.value,'ID')    # Check for reserved words
    return t


def t_newline(t):
    r'\n+'
    t.lexer.lineno += len(t.value)


def t_error(t):
    print("Illegal character '%s'" % t.value[0])
    t.lexer.skip(1)


lexer = lex.lex()
```

Parser File (yacc.py)

```
import ply.yacc as yacc
import sys
import bottlecap

tokens = bottlecap.tokens

OpPrecedence = (

    ('left','ADD','SUB'),
    ('left','MUL','DIV')
)

ExprPrecedence = (

    ('left','PLUS','MINUS'),
    ('left','TIMES','DIVIDE')
)

def p_program(p):

    '''
    program : statements

    '''
    p[0] = p[1]

def p_Sstatement(p):
    '''
    statements : statement
    '''

    p[0] = p[1]


def p_Mstatements(p):
   '''
   statements : statement NEXT statements
   '''
   p[0] = p[3]
```

```
def p_statement(p):
    '''
    statement : assignment_statement
              | operation_statement
              | print_statement
              | expression

    '''
    p[0] = p[1]

env = {}

def p_assignment_statement(p):
    '''
    assignment_statement : ASSIGN OBRACE ID CBRACE TO OBRACE expression CBRACE
    '''
    env[p[3]] = p[7]
    p[0] = p[7]



def p_print_statement(p):
    '''
    print_statement : PRINT OBRACE expression CBRACE

    '''
    p[0] = p[3]
    print(p[0])

def p_operation_statement(p):
    '''
    operation_statement : MUL OBRACE statement  CBRACE BY OBRACE statement CBRACE
                        | DIV OBRACE statement  CBRACE BY OBRACE statement CBRACE
                        | ADD OBRACE statement  CBRACE W OBRACE statement CBRACE
                        | SUB OBRACE statement  CBRACE FROM OBRACE statement CBRACE

    '''

    if p[1] == 'Multiply':
        p[0] = p[3] * p[7]
    elif p[1] == 'Divide':
        p[0] = p[3] / p[7]
```

```
    elif p[1] == 'Add':
        p[0] = p[3] + p[7]
    elif p[1] == 'Subtract':
        p[0] = p[3] - p[7]


def p_expression(p):
    '''

    expression : expression expression TIMES
               | expression expression DIVIDE
               | expression expression PLUS
               | expression expression MINUS


    '''
    if p[3] == '*':
         p[0] = p[1] * p[2]
    elif p[3] == '/':
         p[0] = p[1] / p[2]
    elif p[3] == '+':
         p[0] = p[1] + p[2]
    elif p[3] == '-':
         p[0] = p[1] - p[2]


def p_expression_variable(p):
    '''
    expression : ID

    '''
    if p[1] not in env:
        print("Undeclared Variable.")
    else:
        p[0] = env[p[1]]



def p_expression_number(p):
    '''
    expression : NUMBER

    '''
```

```
    p[0] = p[1]


def p_error(p):
    print("Syntax Error Found.")


parser = yacc.yacc()

while True:
    try:
        s = input('< ')
    except EOFError:
        break
    if not s:
        continue

    result = parser.parse(s)
    print(result)
```