P.3

3 LEX 과제

#1 위의 프로그램 실습해 보기

Pattern-Matching.py - C:/Users/MINSU/Desktop/Pattern-Matching.py (3.6.5)

```python
from ply.lex import lex
tokens = [ 'NUM1', 'NUM2', 'NUM3', 'EQ', 'ASSGN' ]
literals = ['*', '+', '-' ]
t_ignore = ' \t\n'
# Token specifications (as regexs)
t_NUM1 = r'[0-5]+'
t_NUM2 = r'[0-9]+' #r'\d+'
# t_NUM3 = r'([0-9])+'
t_EQ = r'='
t_ASSGN = r'=='
lexer = lex()
data = '''
12345 26 + 8 * 10 + -207 *2 = ==
'''
lexer.input(data)
while True:
    tok = lexer.token()
    if not tok:
        break  # No more input
    print(tok)
```

Python 3.6.5 Shell

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64
bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
============ RESTART: C:/Users/MINSU/Desktop/Pattern-Matching.py ======
======
WARNING: No t_error rule is defined
LexToken(NUM1,'12345',1,1)
LexToken(NUM1,'2',1,7)
LexToken(NUM2,'6',1,8)
LexToken(+,'+',1,10)
LexToken(NUM2,'8',1,12)
LexToken(*,'*',1,14)
LexToken(NUM1,'10',1,16)
LexToken(+,'+',1,19)
LexToken(-,'-',1,21)
LexToken(NUM1,'20',1,22)
LexToken(NUM2,'7',1,24)
LexToken(*,'*',1,26)
LexToken(NUM1,'2',1,27)
LexToken(EQ,'=',1,29)
LexToken(ASSGN,'==',1,31)
>>>
```

#2 HEX

2-ply_ex.py - C:/Users/MINSU/Desktop/2-ply_ex.py (3.6.5)

```python
# plyexample.py
#
# Example of parsing with PLY

from ply.lex import lex

# Token list
tokens = [ 'NUM', 'PLUS', 'MINUS', 'TIMES', 'DIVIDE', 'LPAREN', 'RPAREN', 'ID', 'HEX']

# Ignored characters

t_ignore = ' \t\n'

# Token specifications (as regexs)
t_PLUS   = r'\+'
t_MINUS  = r'-'
t_TIMES  = r'\*'
t_DIVIDE = r'/'
t_LPAREN = r'\('
t_RPAREN = r'\)'

# Token processing functions
def t_HEX(t):
    r'[0][x][0-9a-f]+'
    t.value = int(t.value, 16)
    return t

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

예를들어 0x8a의 경우 NUM이나 ID로 인식되기 전에 미리 HEX로 인식하기 위해 HEX 함수의 위치를 상단으로 설정

16진수를 10진수로 변환

2-ply_ex.py - C:/Users/MINSU/Desktop/2-ply_ex.py (3.6.5)

```
    return t

def t_ID(t):
    r'[a-zA-Z][a-zA-Z_0-9]*'
    # define a Symbol Table
    return t



# Error handler
def t_error(t):
    print('Bad character: {!r}'.format(t.value[0]))
    t.skip(1)

# Build the lexer
lexer = lex()

# Test it out
data = '''
3 + 4 * 10  + -20 *2  0x10 0x1f 0x1a
'''

# Give the lexer some input
lexer.input(data)

# Tokenize
while True:
    tok = lexer.token()
    if not tok:
        break      # No more input
    print(tok)
```

실행결과

2-ply_ex.py - C:/Users/MINSU/Desktop/2-ply_ex.py (3.6.5)

```
# plyexample.py
#
# Example of parsing with PLY

from ply.lex import lex

# Token list
tokens = [ 'NUM', 'PLUS', 'MINUS', 'TIMES', 'DIVIDE', 'LPAREN', 'RPAREN'

# Ignored characters

t_ignore = ' \t\n'

# Token specifications (as regexs)
t_PLUS   = r'\+'
t_MINUS  = r'-'
t_TIMES  = r'\*'
t_DIVIDE = r'/'
t_LPAREN = r'\('
t_RPAREN = r'\)'

# Token processing functions
def t_HEX(t):
    r'[0][x][0-9a-f]+'
    t.value = int(t.value, 16)
    return t

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

Python 3.6.5 Shell

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64
bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:/Users/MINSU/Desktop/2-ply_ex.py ==========
======
LexToken(NUM,3,1,1)
LexToken(PLUS,'+',1,3)
LexToken(NUM,4,1,5)
LexToken(TIMES,'*',1,7)
LexToken(NUM,10,1,9)
LexToken(PLUS,'+',1,13)
LexToken(MINUS,'-',1,15)
LexToken(NUM,20,1,16)
LexToken(TIMES,'*',1,19)
LexToken(NUM,2,1,20)
LexToken(HEX,16,1,23)
LexToken(HEX,31,1,28)
LexToken(HEX,26,1,33)
>>>
```

5 YACC 과제

#3 프로그램 코드 이해하기

-exp_parsing_only.py

exp_parsing_only.py - C:/Users/MINSU/Desktop/Python/계산이론/exp_parsing_only.py (3.6.5)

```python
# ---------------------------------------------------------------------------
# Parsing only : Syntax Check
#   - ex) x = x + y   (semantic error) is ok syntactically
#   - no sign is displayed when parsing is done successfully
# ---------------------------------------------------------------------------

tokens = (
    'NAME','NUMBER',
    'PLUS','MINUS','TIMES','DIVIDE','EQUALS',
    'LPAREN','RPAREN',
    )

# Tokens

t_PLUS    = r'\+'
t_MINUS   = r'-'
t_TIMES   = r'\*'
t_DIVIDE  = r'/'
t_EQUALS  = r'='
t_LPAREN  = r'\('
t_RPAREN  = r'\)'
t_NAME    = r'[a-zA-Z_][a-zA-Z0-9_]*'

def t_NUMBER(t):
    r'\d+'
    try:
        t.value = int(t.value)
    except ValueError:
        print("Integer value too large %d", t.value)
        t.value = 0
```

Python 3.6.5 Shell

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64
bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
====== RESTART: C:/Users/MINSU/Desktop/Python/계산이론/exp_parsing_only
.py ======
Generating LALR tables
calc > 1+1
calc > x+3
calc > 5x+1
Syntax error at 'x'
calc > 15+*10
Syntax error at '*'
calc >
```

-exp_interpreter.py

exp_interpreter.py - C:/Users/MINSU/Desktop/Python/계산이론/exp_interpreter.py (3.6.5)

```python
# ---------------------------------------------------------------------------
# Interpreter for expressions with variables
#  1. Lexing with regular expression
#  2. Parsing with context-free grammar
#  3. Interpretation and evaluation
#  ex)  x = x + y   is an error since "x" and "y" are undefined
# ---------------------------------------------------------------------------

tokens = (
    'NAME','NUMBER',
    'PLUS','MINUS','TIMES','DIVIDE','EQUALS',
    'LPAREN','RPAREN',
    )

# Tokens

t_PLUS    = r'\+'
t_MINUS   = r'-'
t_TIMES   = r'\*'
t_DIVIDE  = r'/'
t_EQUALS  = r'='
t_LPAREN  = r'\('
t_RPAREN  = r'\)'
t_NAME    = r'[a-zA-Z_][a-zA-Z0-9_]*'

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

Python 3.6.5 Shell

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64
bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
======= RESTART: C:/Users/MINSU/Desktop/Python/계산이론/exp_interpreter
.py =======
calc > 10+15
25
calc > x=3
calc > y=5
calc > x*y
15
calc > x*+y
Syntax error at '+'
5
calc >
```

-exp_parsing_ast.py

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64
bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
======= RESTART: C:/Users/MINSU/Desktop/Python/계산이론/exp_parsing_ast
.py =======
('+', ('V', 10), ('*', ('V', 20), ('V', 3)))
70
>>>
======= RESTART: C:/Users/MINSU/Desktop/Python/계산이론/exp_parsing_ast
.py =======
('+', ('-', ('V', 10), ('*', ('+', ('V', 2), ('V', 3)), ('V', 3))), ('V
', 10))
5
>>>
======= RESTART: C:/Users/MINSU/Desktop/Python/계산이론/exp_parsing_ast
.py =======
('+', ('*', ('V', 2), ('UMINUS', ('+', ('V', 1), ('V', 2)))), ('V', 2))
-4
>>>
======= RESTART: C:/Users/MINSU/Desktop/Python/계산이론/exp_parsing_ast
.py =======
('=', 'x', ('V', 10))
None
>>>
======= RESTART: C:/Users/MINSU/Desktop/Python/계산이론/exp_parsing_ast
.py =======
('=', 'x', ('*', ('+', ('V', 1), ('V', 2)), ('V', 3)))
None
>>>
======= RESTART: C:/Users/MINSU/Desktop/Python/계산이론/exp_parsing_ast
```

-exp_eval_ast.py

```
    else:
        print("Syntax error at EOF")

import ply.yacc as yacc
yacc.yacc()

##############  Test Code  #################
prog_exp1 = ''' 10 + 20 * 3 '''             # 70
prog_exp2 = ''' 10 - (2 + 3) * 3 + 10'''    # 5
prog_exp3 = ''' 2 * -(1 + 2) + 2'''         # -4
prog_assign1 = ''' x = 10 '''               # x = 10
prog_assign2 = '''x = (1+2) * 3 '''         # print(names) = {'x' : 9}
prog_exp4 = '''x = x + 10 '''               # x = x + 10
# The Lvalue is different from the Rvalue   #
#############################################

# lexer.input(data)

ast = yacc.parse(prog_exp4)
print(ast)

# dictionary of names
names = {'x' : 9}

def eval(tree):
    if not tree:          return 0      # tree == None
    opr = tree[0]
    if opr == '+':        return eval(tree[1]) +  eval(tree[2])
    elif opr == '-':      return eval(tree[1]) - eval(tree[2])
    elif opr == '*':      return eval(tree[1]) *  eval(tree[2])
    elif opr == '/':      return eval(tree[1]) /  eval(tree[2])
```

```
======= RESTART: C:/Users/MINSU/Desktop/Python/계산이론/exp_parsing_ast
.py =======
('+', ('V', 10), ('*', ('V', 20), ('V', 3)))
70
>>>
======= RESTART: C:/Users/MINSU/Desktop/Python/계산이론/exp_parsing_ast
.py =======
('+', ('-', ('V', 10), ('*', ('+', ('V', 2), ('V', 3)), ('V', 3))), ('V
', 10))
5
>>>
======= RESTART: C:/Users/MINSU/Desktop/Python/계산이론/exp_parsing_ast
.py =======
('+', ('*', ('V', 2), ('UMINUS', ('+', ('V', 1), ('V', 2)))), ('V', 2))
-4
>>>
======= RESTART: C:/Users/MINSU/Desktop/Python/계산이론/exp_parsing_ast
.py =======
('=', 'x', ('V', 10))
None
>>>
======= RESTART: C:/Users/MINSU/Desktop/Python/계산이론/exp_parsing_ast
.py =======
('=', 'x', ('*', ('+', ('V', 1), ('V', 2)), ('V', 3)))
None
>>>
======= RESTART: C:/Users/MINSU/Desktop/Python/계산이론/exp_parsing_ast
.py =======
('=', 'x', ('+', ('NAME', 'x'), ('V', 10)))
None
>>>
```

-exp_simple.py

exp_simple.py - C:/Users/MINSU/Desktop/Python/계산이론/exp_simple.py (3.6.5)

```python
# ---------------------------------------------------------------------
# Interpreter for expressions without variables
#  No ambigous grammars, so precedence is not defined
#  1. Lexing with regular expression
#  2. Interpretation and evaluation
# ---------------------------------------------------------------------

from ply.lex import lex
from ply.yacc import yacc

# Token list
tokens = [ 'NUM', 'PLUS', 'MINUS', 'TIMES', 'DIVIDE', 'LPAREN', 'RPAREN'

# Ignored characters

t_ignore = ' \t\n'

# Token specifications (as regexs)
t_PLUS   = r'\+'
t_MINUS  = r'-'
t_TIMES  = r'\*'
t_DIVIDE = r'/'
t_LPAREN = r'\('
t_RPAREN = r'\)'

# Token processing functions
def t_NUM(t):
    r'\d+'
    t.value = int(t.value)
    return t
```

Python 3.6.5 Shell

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64
bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========= RESTART: C:/Users/MINSU/Desktop/Python/계산이론/exp_simple.py
=========
2
5
37
>>>
```

#4 파일 eval_ast 에서 NUMBER->N

Untitled

```python
# ---------------------------------------------------------------------
# Evaluator of Abstract Syntax Trees
# Sequence of statements with  ";"
# ---------------------------------------------------------------------


#######################  Test Code  ############################
ast1 = ('+', ('N',10), ('N',20))    # 10 + 20

# x = 10 + 20
ast2 = ('=', 'x',
             ('+', ('N',10), ('N',20))
        )

# x = 10 + 20 ; x + x
ast3 = (';', ('=', 'x', ('+', ('N',10), ('N',20))),
             ('+', ('NAME','x'), ('NAME','x'))
        )
#################################################################

ast = ast3

print(ast)

# dictionary of names
names = {}

def eval(tree):
    if not tree:          return 0      # tree == None
    opr = tree[0]
    if opr == '+':        return eval(tree[1]) +  eval(tree[2])
```

Untitled

```python
        )

# x = 10 + 20 ; x + x
ast3 = (';', ('=', 'x', ('+', ('N',10), ('N',20))),
             ('+', ('NAME','x'), ('NAME','x'))
        )
#################################################################

ast = ast3

print(ast)

# dictionary of names
names = {}

def eval(tree):
    if not tree:          return 0      # tree == None
    opr = tree[0]
    if opr == '+':        return eval(tree[1]) +  eval(tree[2])
    elif opr == '-':      return eval(tree[1]) - eval(tree[2])
    elif opr == '*':      return eval(tree[1]) *  eval(tree[2])
    elif opr == '/':      return eval(tree[1]) /  eval(tree[2])
    elif opr == 'UMINUS': return - eval(tree[1])
    elif opr == 'N': return tree[1]
    elif opr == 'NAME':   return names[tree[1]]
    elif opr == ';':      eval(tree[1]); return eval(tree[2])
    elif opr == '=':      names[tree[1]] = eval(tree[2])
    else:                 print("unexpeced case : ", tree)

print(eval(ast))
```

#5 (1+21)*(3+4)

eval_ast.py - C:/Users/MINSU/Desktop/eval_ast.py (3.6.5)

File Edit Format Run Options Window Help

```python
# ---------------------------------------------------------------------------
# Evaluator of Abstract Syntax Trees
# Sequence of statements with  ";"
# ---------------------------------------------------------------------------


######################  Test Code  ############################
ast1 = ('+', ('N', 10), ('N', 20))  # 10 + 20

# x = 10 + 20
ast2 = ('=', 'x',
        ('+', ('N', 10), ('N', 20))
        )

# x = 10 + 20 ; x + x
ast3 = (';', ('=', 'x', ('+', ('N', 10), ('N', 20))),
        ('+', ('NAME', 'x'), ('NAME', 'x'))
        )
#################################################################
# x = (1 + 21) * (3 + 4)
ast4 = ('*',
        ('+', ('N', 1), ('N', 21)),
         ('+', ('N', 3), ('N', 4)))

ast = ast4

print(ast)

# dictionary of names
names = {}
```

Ln: 1 Col: 0

밑에 코드는 기존과 동일

eval_ast.py - C:/Users/MINSU/Desktop/eval_ast.py (3.6.5)

File Edit Format Run Options Window Help

```python
# dictionary of names
names = {}


def eval(tree):
    if not tree:          return 0  # tree == None
    opr = tree[0]
    if opr == '+':
        return eval(tree[1]) + eval(tree[2])
    elif opr == '-':
        return eval(tree[1]) - eval(tree[2])
    elif opr == '*':
        return eval(tree[1]) * eval(tree[2])
    elif opr == '/':
        return eval(tree[1]) / eval(tree[2])
    elif opr == 'UMINUS':
        return - eval(tree[1])
    elif opr == 'N':
        return tree[1]
    elif opr == 'NAME':
        return names[tree[1]]
    elif opr == ';':
        eval(tree[1]); return eval(tree[2])
    elif opr == '=':
        names[tree[1]] = eval(tree[2])
    else:
        print("unexpeced case : ", tree)


print(eval(ast))
```

Ln: 22 Col: 18

실행시 아래와 같이나옴

eval_ast.py - C:/Users/MINSU/Desktop/eval_ast.py (3.6.5)

```python
# ---------------------------------------------------------------------
# Evaluator of Abstract Syntax Trees
# Sequence of statements with  ";"
# ---------------------------------------------------------------------


######################  Test Code  ############################
ast1 = ('+', ('N', 10), ('N', 20))  # 10 + 20

# x = 10 + 20
ast2 = ('=', 'x',
        ('+', ('N', 10), ('N', 20))
        )

# x = 10 + 20 ; x + x
ast3 = (';', ('=', 'x', ('+', ('N', 10), ('N', 20))),
        ('+', ('NAME', 'x'), ('NAME', 'x'))
        )
#################################################################
# x = (1 + 21) * (3 + 4)
ast4 = ('*',
        ('+', ('N', 1), ('N', 21)),
         ('+', ('N', 3), ('N', 4)))

ast = ast4

print(ast)

# dictionary of names
names = {}
```

Python 3.6.5 Shell

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64
bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:/Users/MINSU/Desktop/eval_ast.py ==========
======
('*', ('+', ('N', 1), ('N', 21)), ('+', ('N', 3), ('N', 4)))
154
>>>
```

5.1 변수의 사용

#6 x*x+x

ast5.py - C:/Users/MINSU/Desktop/ast5.py (3.6.5)

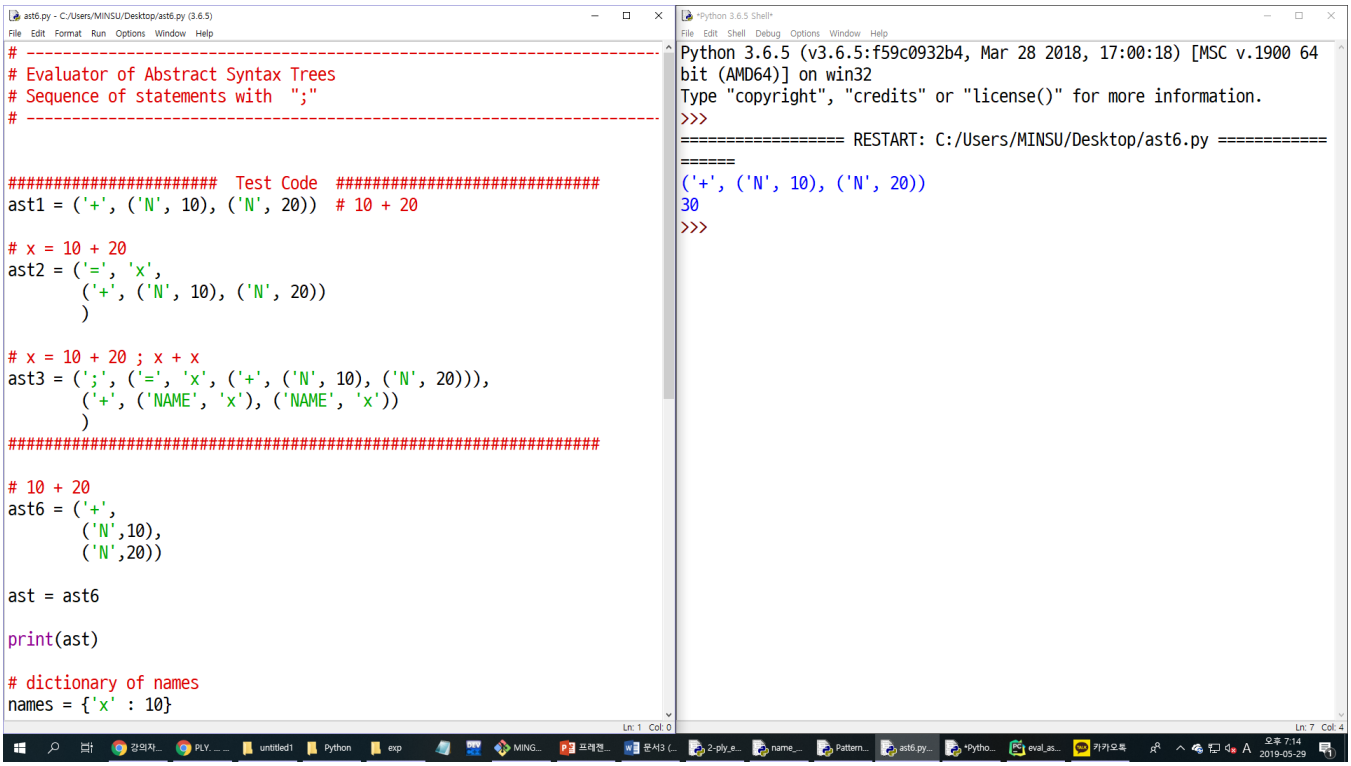
```python
# ---------------------------------------------------------------------
# Evaluator of Abstract Syntax Trees
# Sequence of statements with  ";"
# ---------------------------------------------------------------------


######################  Test Code  ############################
ast1 = ('+', ('N', 10), ('N', 20))  # 10 + 20

# x = 10 + 20
ast2 = ('=', 'x',
        ('+', ('N', 10), ('N', 20))
        )

# x = 10 + 20 ; x + x
ast3 = (';', ('=', 'x', ('+', ('N', 10), ('N', 20))),
        ('+', ('NAME', 'x'), ('NAME', 'x'))
        )
#################################################################

# x * x + x
ast5 = ('+',
        ('NAME', 'x'),
        ('*', ('NAME', 'x'), ('NAME', 'x')))

ast = ast5

print(ast)

# dictionary of names
names = {'x' : 10}
```

Python 3.6.5 Shell

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64
bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: C:/Users/MINSU/Desktop/ast5.py ============
======
('+', ('NAME', 'x'), ('*', ('NAME', 'x'), ('NAME', 'x')))
Traceback (most recent call last):
  File "C:/Users/MINSU/Desktop/ast5.py", line 59, in <module>
    print(eval(ast))
  File "C:/Users/MINSU/Desktop/ast5.py", line 38, in eval
    return eval(tree[1]) + eval(tree[2])
  File "C:/Users/MINSU/Desktop/ast5.py", line 50, in eval
    return names[tree[1]]
KeyError: 'x'
>>>
================= RESTART: C:/Users/MINSU/Desktop/ast5.py ============
======
('+', ('NAME', 'x'), ('*', ('NAME', 'x'), ('NAME', 'x')))
110
>>>
```

5.2 eval 함수 수행과정

#7 eval(('+', ('N',10), ('N',20))) ast6



#8 x*x+x 과정을 보여라

eval(('NAME', x)) + eval(('NAME', x)) + eval(('NAME', x))

6.1 과제

-LEX단계 발생 스트링 예, 에러발생 이유

```
calc > 'dj'+'ab'
Illegal character '''
Illegal character '''
Illegal character '''
Illegal character '''
```

토큰을 자르는 과정에서 오류가 날 경우 에러가 발생한다.

-YACC단계 발생 스트링 예, 에러발생 이유

12345 26 + 8 * 10 + -207 *2 = ==

5와 2 사이가 떨어져서 숫자와 숫자 사이에 부호가 없기 때문에 계산이 안되고

+뒤에 –가 오는 경우도 따로 –에 관하여 정의를 하지 않으면 오류가 난다.

-ACTION단계 발생 스트링 예, 에러발생 이유(의미에러)

X*x/x

우선순위의 혼동이 발생하는 스트링은 트리가 2개이상 나오는 경우