Analizor lexical folosind generatoare de scannere

C-Minus-Minus Project

2022

Student: **Draghici Andreea-Maria**
Grupa: **CR4.S1A**
Anul de studiu: **IV**
Specializarea: **Calculatoare Romana**

Pentru a gestiona cuvintele rezervate, scriem o singura regula care sa se potriveasca cu un identificator ca mai jos:

```
"""
  Created by: Andreea-Maria Draghici
  November 2022 , CR4 S1A
  C-- Project
"""

import ply.lex as lex  # import the lex from ply library

writer = open("myOutFile.txt", "w")  # represents in which file I will write

# represents the reserved keywords for C-- project
reserved_keywords = {
    'int' : 'INT',
    'main' : 'MAIN',
    'if' : 'IF',
    'else' : 'ELSE',
    'return' : 'RETURN',
    'void' : 'VOID',
    'while' : 'WHILE',
    'cin' : 'CIN',
    'cout' : 'COUT',
    'bool' : 'BOOL',
    'true' : 'TRUE',
    'false' : 'FALSE'
}
```

Definesc o lista ce contine toate numele posibile de token-uri care pot fi produse de catre lexer:

```
# list of token names for C-- project. This is always required
tokens = ['INTCONST', 'IDENT', 'PLUS', 'MULTIPLICATION', 'EQ', 'LPAREN', 'RPAREN', 'LBRACE', 'RBRACE', 'COMMA',
          'SEMICOLON', 'EQEQ', 'NEQ', 'LT', 'LTE', 'GT', 'GTE', 'MINUS', 'DIVISION', 'COMMENT', 'AND', 'LSQBKT',
          'RSQBKT', 'OR', 'UNARY', 'LSHIFT', 'RSHIFT', 'DOT'] + list(reserved_keywords.values())
```

Fiecare token din lista de mai sus este specificat prin scrierea unei reguli de expresie regulata. Iar fiecare regula este definite printr-un prefix special ‘ t_’ pentru a indica ca defineste un token.

```
# regular expression rules for above tokens
t_PLUS = r'\+'
t_MULTIPLICATION = r'\*'
t_EQ = r'='
t_LPAREN = r'\('
t_RPAREN = r'\)'
t_LBRACE = r'\{'
t_RBRACE = r'\}'
t_COMMA = r','
t_SEMICOLON = r';'
t_EQEQ = r'=='
t_NEQ = r'!='
t_LT = r'<'
t_GT = r'>'
t_LTE = r'<='
t_GTE = r'>='
t_MINUS = r'\-'
t_DIVISION = r'\/'
t_COMMENT = r'(/\*(.|\n)*?\*/)|(//.*)|(\#)'  # C-- comment , support the cases: /**/ or // or # or /***/
t_AND = r'\&+'  # support cases: & or && or more
t_OR = r'\|+'  # support cases: | or || or more
t_LSQBKT = r'\['
t_RSQBKT = r'\]'
t_UNARY = r'\_'
t_LSHIFT = r'<<'
t_RSHIFT = r'>>'
t_DOT = r'\.'
```

Numele care urmeaza dupa prefixul t_ trebuie sa se potriveasca exact cu unul dintre numele furnizate de token-uri. O regula de simbol poate fi specificata ca functie. Aceasta regula se potriveste cu numere si converteste sirul intr-un numar intreg.

```
# regular expression rules with action code
# represents the method in which INTCONST will be checked based on the specific regex
def t_INTCONST(t) :
    r"""\d+"""
    t.value = int(t.value)  # check for int constant
    return t  # return token
```

Aceasta regula se potriveste cu text si conversteste string-ul intr-un identificator. Va fi verificat identificatorul in functie de o expresie regulate specificata mai jos.

```
# represents the method in which IDENTIFICATOR will be checked based on the specific regex
def t_IDENT(t) :
    r"""[a-zA-Z][a-zA-Z0-9]*"""
    t.value = reserved_keywords.get(t.value, 'ID')  # check for reserved words
    return t  # return token
```

Pentru a actualiza informatiile ce constituie o ‘linie’ de intrare (Ex: newline) scriem o regula ca mai jos unde verificam lungimea linie si facem update la numarul de linii.

```
# special rule for updating line numbers (lineno)
def t_newline(t) :
    r"""\n+"""
    t.lexer.lineno += len(t.value)  # check the line length and update the line numbers
```

Fortam un symbol sa fie ignorat. Functia t_error() este utilizata pentru a gestiona erorile de lexing care apar atunci cand sunt detectate caractere ilegale / invalide.

```
# characters to be ignored (special rule)
t_ignore = ' \t\n'


# error handling rule (special rule)
def t_error(t) :
    writer.write("INVALID CHARACTER '%s'" % t.value[0] + '\n')
    t.lexer.skip(1)  # for bad characters, we just skip over it
```

In metode test_lexer(), imi construiesc lexer-ul, dupa ce construiesc lexer-ul, utilizez doua metode pentru a controla lexer-ul, mai exact metode input pentr a reseta lexer-ul ce stocheaza continutul de intrare si metoda token ce returneaza urmatorul simbol. Returneaza o instant LexToken speciala.

```python
# test it output
def test_lexer() :
    lexer = lex.lex()  # build the lexer
    file_name = input('ADD THE INPUT FILE NAME: ')

    with open(file_name) as f :  # read the input file

        writer.write(
            "------------ C Minus Minus LEXICAL ANALYZER USING SCANNER GENERATORS ------------\n\n")  # interactive message

        contents = f.read()  # read contents of input file
        writer.write(
            '------------ PRINT ALL CONTENTS OF INPUT FILE ------------ \n' + contents + '\n\n')  # interactive message
        lexer.input(contents)  # give the lexer some input

        writer.write('------------ LEXER TOKENS ARE ------------ \n\n')  # interactive message
        while True :
            tok = lexer.token()  # tokenize
            if not tok :
                break  # no more input , end of file
            writer.write(tok.__str__() + '\n')  # write the tokens into output file

    writer.close()  # close the file
```

Apelez metoda de mai sus pentru a putea rula programul.

```python
# main driver
if __name__ == '__main__' :
    try :
        test_lexer()  # call the test_lexer method and try it out
    except Exception as exception :
        print(f"Filed to running the application due to: {exception}\n")  # print the exception message
```

Rezultatul obtinut in urma rularii codului pe unul dintre testele de input este urmatorul:

main.py | test4.txt | myOutFile.txt | test2.txt | test3.txt | test1.txt

```
------------ C Minus Minus LEXICAL ANALYZER USING SCANNER GENERATORS ------------

------------ PRINT ALL CONTENTS OF INPUT FILE ------------
int main(){

cout<<"This is a sample test.";

return 0;
}

------------ LEXER TOKENS ARE ------------

LexToken(IDENT,'INT',1,0)
LexToken(IDENT,'MAIN',1,4)
LexToken(LPAREN,'(',1,8)
LexToken(RPAREN,')',1,9)
LexToken(LBRACE,'{',1,10)
LexToken(IDENT,'COUT',1,13)
LexToken(LSHIFT,'<<',1,17)
INVALID CHARACTER '"'
LexToken(IDENT,'ID',1,20)
LexToken(IDENT,'ID',1,25)
LexToken(IDENT,'ID',1,28)
LexToken(IDENT,'ID',1,30)
LexToken(IDENT,'ID',1,37)
LexToken(DOT,'.',1,41)
INVALID CHARACTER '"'
LexToken(SEMICOLON,';',1,43)
LexToken(IDENT,'RETURN',1,46)
LexToken(INTCONST,0,1,53)
LexToken(SEMICOLON,';',1,54)
LexToken(RBRACE,'}',1,56)

```