Universidade Federal de Roraima
Centro de Ciência e Tecnologia
Departamento de Ciência da Computação
Disciplina de Compiladores
Prof.: Dr. Luciano Ferreira
Aluno: Felipe Derkian de Sousa Freitas

Analisador Léxico

Boa Vista, 21 de Março de 2020

O que é um Analisador Léxico?

Análise léxica é o processo de analisar a entrada de linhas de caracteres (tal como o código fonte de um programa de computador) e produzir uma sequência de símbolos chamado "símbolos léxicos" (lexical tokens), ou somente "símbolos" (tokens), que podem ser manipulados mais facilmente por um parser (leitor de saída). O componente do compilador responsável pela execução desse processo é conhecido como Analisador léxico.

O analisador léxico, ou scanner como também é chamado, faz a varredura do programa-fonte caractere por caractere e, traduz em uma sequência de símbolos léxicos ou tokens. É nessa fase que são reconhecidas as palavras reservadas, constantes, identificadores e outras palavras que pertencem a linguagem de programação. O analisador léxico executa outras tarefas como por exemplo o tratamento de espaços, eliminação de comentários, contagem do número de linhas que o programa possui e etc.

Algoritmo 3.1 Algoritmo do analisador léxico.

```
SCANNER(M, σ)
 1  s ← ESTADO-INICIAL(M)
 2  while true
 3  do if ISEMPTY(σ)
 4        then if ESTADO-FINAL(M, s)
 5                then return true
 6                else return false
 7        else c ← REMOVEFIRST(σ)
 8             s ← PRÓXIMO-ESTADO(M, s, c)
 9             if s = NIL
10                then return false
```

A análise léxica é a forma de verificar determinado alfabeto. Quando analisamos uma palavra, podemos definir através da análise léxica se existe ou não algum caractere que não faz parte do nosso alfabeto, ou um alfabeto inventado por nós.

É a primeira etapa do processo de compilação e seu objetivo é dividir o código fonte em símbolos, preparado-o para a Análise Sintática. Neste processo pode-se destacar três atividades como fundamentais:

- Extração e classificação dos tokens;
- Eliminação de delimitadores e comentários;
- Recuperação de Erros.

O analisador léxico funciona de duas maneiras:

- Primeiro estado da análise: A primeira etapa lê a entrada de caracteres, um de cada vez, mudando o estado em que os caracteres se encontram. Quando o analisador encontra um caracter que ele não identifica como correto, ele o chama de "estado morto" então, ele volta à última análise que foi aceita e assim tem o tipo e comprimento do léxico válido.

Um léxico, entretanto, é uma única lista de caracteres conhecidas de ser um tipo correto. Para construir um símbolo, o analisador léxico necessita de um segundo estado.

• Segundo estado da análise: Nesta etapa são repassados os caracteres do léxico para produzir um valor. O tipo do léxico combinado com seu valor é o que adequadamente constitui um símbolo, que pode ser dado a um parser.

A análise léxica escreve um parser muito mais fácil. Em vez de ter que acumular, renomeia seus caracteres individualmente. O parser não mais se preocupa com símbolos e passa a preocupar-se só com questões de sintática. Isto leva a eficiência de programação, e não eficiência de execução. Entretanto, desde que o analisador léxico é o subsistema que deve examinar cada caracter único de entrada, podem ser passos intensivos e o desempenhos se torna crítico, pode estar usando um compilador.

As desvantagens da Análise Léxica são o tratamento de dados em branco, formato fixo de entrada e a inexistência de palavras reservadas, em determinadas linguagens.

Analisador Léxico feito em Python 3.6

O analisador léxico a seguir lê um arquivo de texto chamado ‘cod.txt’ e realiza a remoção de tabulações e quebras de linhas. Depois procura pelo símbolo de (;) fim de instrução e adiciona um espaço para depois realizar um split e separar as palavras de acordo com sua representação em palavras reservadas, operadores, atribuidores, entre outros como pode ser visto no código abaixo.

Código em Python

```
#palavras reservadas
reserved_words = ['while', 'for', 'do', 'return', 'break', 'continue', 'switch', 'case']

#operadores
operators = ['+', '-', '/', '*', '^']

#atribuidores
assigners = ['=', '+=', '-=', '/=', '*=']

#valida int
def isInt(value):
  try:
    int(value)
    return True
```

```
  except:
    return False

#valida float
def isFloat(value):
  try:
    float(value)
    return True
  except:
    return False



def content_file_print(content):
    print('')
    print("File Contente: ")
    print('>>>\n{}'.format(content))
    print('>>>\n')

def print_lexemas(lexema):
    print("Lexemas:")
    print(lexema)

def print_code_help(code):
    print("code help:\n{}\n".format(code))

def add_espace_final_instruct(code):
    #add espace to ;
    code = code.replace(";", " ;")
    return code

def replace_to_break_line(code):
    #replace \n to espace
```

```
    code = code.replace("\n", " ")
    return code

def replace_to_tabulation(code):
    #replace \t to espace
    code = code.replace("\t", " ")
    return code

def lexemas(code):
    #split code
    list_lexema = code.split()

    #matrix of lexema
    matriz = []

    #para cada palavra examina seu tipo
    for word in list_lexema:
        if word in reserved_words:
            matriz.append([word, word])
        elif word in operators:
            matriz.append([word, word])
        elif word in assigners:
            matriz.append([word, word])
        elif word == ';':
            matriz.append([word, '$'])
        elif isInt(word) or isFloat(word):
            matriz.append([word, 'num'])
        else:
            matriz.append([word, 'id'])
    return matriz

def analise_lexema(code):
```

```
    #functions tratament
    code = add_espace_final_instruct(code)
    code = replace_to_break_line(code)
    code = replace_to_tabulation(code)

    #print codigo auxiliar
    print_code_help(code)

    #funtion get lexemas
    lexema = lexemas(code)

    #print lexemas
    print_lexemas(lexema)

#function main
if 'main' == 'main':
    #open file cod.txt
    arquivo = open('cod.txt', 'r')

    #read file
    code = arquivo.read()
    #type cast
    code = str( code )

    #function print content file
    content_file_print(code)

    #function analise lexema
    analise_lexema(code)

    #close file
    arquivo.close()
```

Casos de teste

Caso de teste com o seguinte código no arquivo:

a = x + 5 * 10;

b = a + c * d;

c = 1000 * 100 + H;

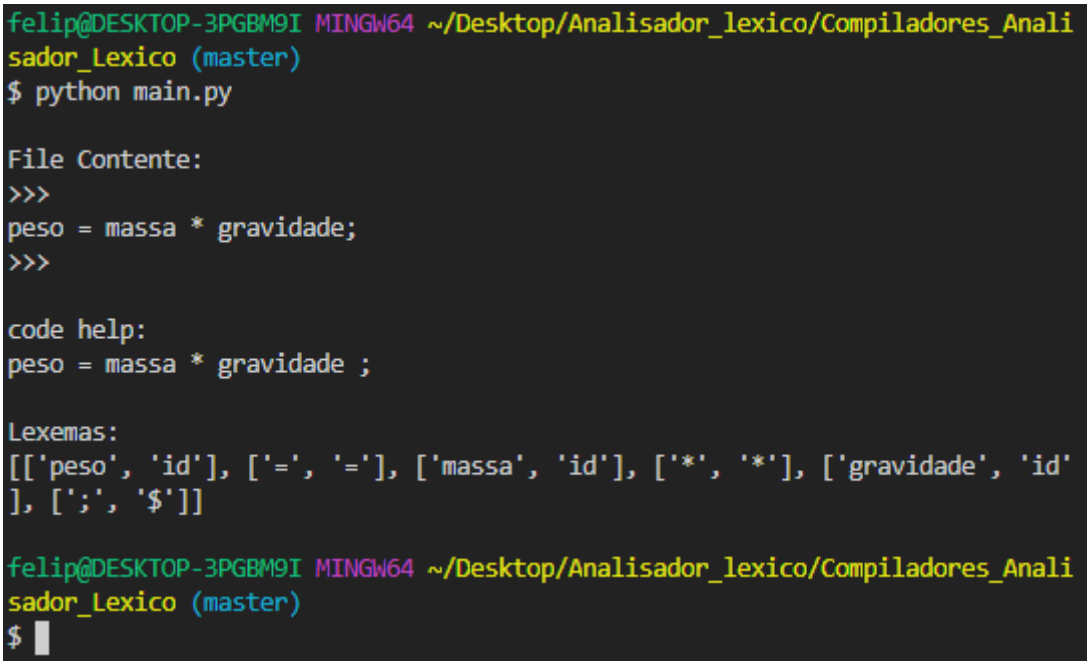
Saída pelo algoritmo:

```
felip@DESKTOP-3PGBM9I MINGW64 ~/Desktop/Analisador_lexico/Compiladores_Analisador_Lexico (master)
$ python main.py

File Contente:
>>>
a = x + 5 * 10;

b = a + c * d;

c = 1000 * 100 + H;
>>>

code help:
a = x + 5 * 10 ;  b = a + c * d ;  c = 1000 * 100 + H ;

Lexemas:
[['a', 'id'], ['=', '='], ['x', 'id'], ['+', '+'], ['5', 'num'], ['*', '*'],
 ['10', 'num'], [';', '$'], ['b', 'id'], ['=', '='], ['a', 'id'], ['+', '+']
, ['c', 'id'], ['*', '*'], ['d', 'id'], [';', '$'], ['c', 'id'], ['=', '='],
 ['1000', 'num'], ['*', '*'], ['100', 'num'], ['+', '+'], ['H', 'id'], [';',
 '$']]

felip@DESKTOP-3PGBM9I MINGW64 ~/Desktop/Analisador_lexico/Compiladores_Analisador_Lexico (master)
$
```

A saída mostra o ‘File Contet’, ou seja, o que foi lido do arquivo. Depois, mostra o ‘code help’ que é o resultado da passagem pelas funções de tratamento do código fonte informado e após isso é mostrando os lexemas em uma matriz mostrado em cada posição a palavra e o tipo como id, num, $, símbolos de operação aritmética, atribuição, entre outros.

Caso de teste com o seguinte código no arquivo:

peso = massa * gravidade;



Caso de teste com o seguinte código no arquivo com fórmula de Baskara:

a , b , c , x1 , x2 , delta;

delta = ((b ^ 2) - (4 * a * c));

x1 = ((- + (delta)) / (2 * a);

x2 = ((- - (delta)) / (2 * a);

```
felip@DESKTOP-3PGBM9I MINGW64 ~/Desktop/Analisador_lexico/Compiladores_Anali
sador_Lexico (master)
$
$ python main.py

File Contente:
>>>
a , b , c , x1 , x2 , delta;
delta = ( ( b ^ 2 ) - ( 4 * a * c ) );
x1 = ( ( - + ( delta ) ) / ( 2 * a );
x2 = ( ( - - ( delta ) ) / ( 2 * a );

>>>

code help:
a , b , c , x1 , x2 , delta ; delta = ( ( b ^ 2 ) - ( 4 * a * c ) ) ; x1 = (
 ( - + ( delta ) ) / ( 2 * a ) ; x2 = ( ( - - ( delta ) ) / ( 2 * a ) ;

Lexemas:
[['a', 'id'], [',', ','], ['b', 'id'], [',', ','], ['c', 'id'], [',', ','],
['x1', 'id'], [',', ','], ['x2', 'id'], [',', ','], ['delta', 'id'], [';', '
$'], ['delta', 'id'], ['=', '='], ['(', '('], ['(', '('], ['b', 'id'], ['^',
 '^'], ['2', 'num'], [')', ')'], ['-', '-'], ['(', '('], ['4', 'num'], ['*',
 '*'], ['a', 'id'], ['*', '*'], ['c', 'id'], [')', ')'], [')', ')'], [';', '
$'], ['x1', 'id'], ['=', '='], ['(', '('], ['(', '('], ['-', '-'], ['+', '+'
], ['(', '('], ['delta', 'id'], [')', ')'], [')', ')'], ['/', '/'], ['(', '(
'], ['2', 'num'], ['*', '*'], ['a', 'id'], [')', ')'], [';', '$'], ['x2', 'i
d'], ['=', '='], ['(', '('], ['(', '('], ['-', '-'], ['-', '-'], ['(', '('],
 ['delta', 'id'], [')', ')'], [')', ')'], ['/', '/'], ['(', '('], ['2', 'num
'], ['*', '*'], ['a', 'id'], [')', ')'], [';', '$']]

felip@DESKTOP-3PGBM9I MINGW64 ~/Desktop/Analisador_lexico/Compiladores_Anali
sador_Lexico (master)
$
```

Referências

https://pt.wikibooks.org/wiki/Constru%C3%A7%C3%A3o_de_compiladores/An%C3%A1lise_l%C3%A9xica

http://www.dca.fee.unicamp.br/cursos/EA876/apostila/HTML/node50.html

https://www.clubedohardware.com.br/forums/topic/857550-formula-de-baskara-no-visualg/

https://www.w3schools.com/python/