Relatório do Trabalho 1 - Construção de Compiladores - INE5426
Grupo: David Grunheidt Vilela Ordine - 16202253

O analisador léxico foi desenvolvido em Python e utiliza a biblioteca PLY. Abaixo estão os itens pedidos na descrição da atividade.

1. Identificação dos tokens:

A identificação dos tokens foi feita tomando como base o livro do Delamaro, especificamente, na seção 3.3. Desta maneira, identificamos os seguintes tokens, os quais serão listados em grupos de funcionalidades para facilitar o entendimento.

A. **Tokens de palavras reservadas:** DEF, NEW, BREAK, FOR, IF, ELSE, INT, FLOAT, STRING, PRINT, READ, RETURN
B. **Tokens de operadores:** ASSIGN, GT, LT, EQ, LE, GE, NEQ, PLUS, MINUS, STAR, SLASH, REM
C. **Tokens de símbolos especiais:** LPAREN, RPAREN, LBRACE, RBRACE, LBRACKET, RBRACKET, SEMICOLON, COMMA, DOT
D. **Tokens de constantes:** INTCONST, FLOATCONST, STRINGCONST, NULLCONST
E. **Token de um identificador:** ID
F. **Tokens de caracteres ignorados:** ignore, newline, COMMENT

2. Definições regulares para cada token:

Foram geradas as seguintes definições regulares para cada um dos tokens citados acima, as quais serão novamente separadas em grupos de funcionalidade. A notação mostrada abaixo segue o padrão NOME_DO_TOKEN: "EXPRESSAO_REGULAR_QUE_O_DEFINE".

A. Expressões regulares dos tokens de palavras reservadas:
   a. DEF: "def"
   b. NEW: "new"
   c. BREAK: "break"
   d. FOR: "for"
   e. IF: "if"
   f. ELSE: "else"
   g. INT: "int"
   h. FLOAT: "float",
   i. STRING: "string",
   j. PRINT: "print",
   k. READ: "read",
   l. RETURN: "return"

B. Expressões regulares dos tokens de operadores:
   a. ASSIGN: "="
   b. GT: ">"
   c. LT: "<"
   d. EQ: "=="
   e. LE: "<="

f. GE: ">="
g. NEQ: "!="
h. PLUS: "+"
i. MINUS: "-"
j. STAR: "*"
k. SLASH: "/"
l. REM: "%"

C. Expressões regulares dos tokens de símbolos especiais:

a. LPAREN: "("
b. RPAREN: ")"
c. LBRACE: "{"
d. RBRACE: "}"
e. LBRACKET: "["
f. RBRACKET: "]"
g. SEMICOLON: ";"
h. COMMA: ","
i. DOT: "."

D. Expressões regulares dos tokens de constantes:

a. INTCONST: "[0-9]+"
    i. Uma ou mais ocorrências de um número de 0 a 9.
b. FLOATCONST: "[0-9]+.[0-9]+"
    i. Uma ou mais ocorrências de um número de 0 a 9, seguido de um ponto, seguido de uma ou mais ocorrências de um número de 0 a 9.
c. STRINGCONST: "\"([^\\\"]|\\.)*\""
    i. Uma ocorrência de ", seguido de zero ou mais ocorrências de qualquer caractere exceto a barra invertida ("\") e " ou de uma barra invertida seguida de qualquer caractere, seguido de uma ocorrência de ".
d. NULLCONST: "null"

E. Expressão regular do token identificador:

a. ID: "[a-zA-Z_][a-zA-Z_0-9]*"
    i. Uma ocorrência de uma letra de "a" até "z" seguido de zero ou mais ocorrências de uma letra de "a" até "z" ou de "A" até "Z" ou um número de 0 até 9.

F. Expressões regulares dos tokens de caracteres ignorados:

a. newline: "\n+"
    i. uma ou mais ocorrências de "\n"”
b. ignore: "(|\t)+"
    i. Uma ou mais ocorrências de um espaço ou um tab.
c. COMMENT: "//.*"
    i. Duas barras seguido de indefinidas ocorrências de qualquer caractere.

5. Descrição da ferramenta utilizada:

Para auxílio do desenvolvimento do trabalho, foi utilizado a biblioteca PLY pra Python. Primeiramente, é necessário importar a biblioteca:

```
import ply.lex as lex
```

Para definir os tokens é necessário instanciar uma lista com todos os nomes destes e instanciar as variáveis que definem cada expressão regular que rege cada token. As variáveis que contêm as expressões regulares devem começar com t_ e em seguida deve conter o nome do token. A lista com os nomes dos tokens deve ser chamada de "tokens". Ex:

```
tokens = ['NUMBER', 'PLUS', 'MINUS']
t_NUMBER = r'[0-9]+'
t_PLUS = r'\+'
t_MINUS = r'-'
```

Também é possível definir regras de tokens que realizem ações através de funções.
Ex:

```
# Define a rule so we can track line numbers
 def t_newline(t):
     r'\n+'
     t.lexer.lineno += len(t.value)
```

No exemplo anterior, vemos que nada é retornado na função. Quando definimos uma regra com uma ação que não retorna um token, diz-se que esse token é um token descartável. Além disso, a variável t_ignore define todos os caracteres que serão ignorados. Diferentemente das regras anteriores, nessa variável é armazenado uma string ao invés de um regex.

Por fim, temos uma regra para lidar com um erro léxico:

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

Após definidos os tokens e suas regras, instanciamos o analisador léxico da seguinte forma:

```
lexer = lex.lex()
```

O analisador ("lexer") recebe um input do tipo string, a qual representa o código a ser analisado.

```
lexer.input(data)
```

Para gerar os tokens, podemos realizar a seguinte iteração sobre o lexer:

```
for tok in lexer:
     print(tok)
```

O que equivale a seguinte implementação:

```
while True:
      tok = lexer.token()
      if not tok:
          break      # No more input
      print(tok.type, tok.value, tok.lineno, tok.lexpos)
```

Essa implementação equivalente mostra a função que retorna um token, "lexer.token()". O Token retornado é do tipo "LexToken" e possui os atributos "tok.type, tok.value, tok.lineno, and tok.lexpos". "Lineno" representa a linha e "pos" a coluna onde aquele lexema se encontra. Os tokens são retornados na mesma ordem em que aparecem no arquivo.

Para visualizar os tokens retornados, segue um resultado quando definimos um simples código.

```
Código:

if (1 == 2) {
      test = 1.122
      test2 = "String é isso \" aqui \n"
}
```

Tokens produzidos:

```
LexToken(IF,'if',1,0)
LexToken(LPAREN,'(',1,3)
LexToken(INTCONST,'1',1,4)
LexToken(EQ,'==',1,6)
LexToken(INTCONST,'2',1,9)
LexToken(RPAREN,')',1,10)
LexToken(LBRACE,'{',1,12)
LexToken(ID,'test',2,15)
LexToken(ASSIGN,'=',2,20)
LexToken(FLOATCONST,'1.122',2,22)
LexToken(ID,'test2',3,29)
LexToken(ASSIGN,'=',3,35)
LexToken(STRINGCONST,'"String é isso \\" aqui \\n"',3,37)
LexToken(RBRACE,'}',4,64)
```

Modificações na linguagem:

Foi alterado a regra PARAMLIST na gramática para o seguinte:

PARAMLIST -> ((int | float | string) ident ([])?, PARAMLIST | (int | float | string) ident ([])?)