01-compiladores-analise-lexica-tpplex

January 4, 2021

Contents

Chapter 1

Análise Léxica

1.1 Preparação do Ambiente

- Instalação do PLY

In [ ]:
```
!pip install ply
```

```
Collecting ply
  Downloading https://files.pythonhosted.org/packages/a3/58/35da89ee790598a0700ea49b2a66594140f44dec458c
     || 51kB 2.4MB/s eta 0:00:011
Installing collected packages: ply
Successfully installed ply-3.11
```

In [ ]:
```
!jupyter nbextension install https://rawgit.com/jfbercher/small_nbextensions/master/highlighter.
!jupyter nbextension enable highlighter/highlighter
```

```
Downloading: https://rawgit.com/jfbercher/small_nbextensions/master/highlighter.zip -> /tmp/tmpxTceyO/hi
Extracting: /tmp/tmpxTceyO/highlighter.zip -> /root/.local/share/jupyter/nbextensions
Enabling notebook extension highlighter/highlighter...
      - Validating: OK
```

In [ ]:
```
%%javascript
require("base/js/utils").load_extensions("highlighter/highlighter")
```

```
<IPython.core.display.Javascript object>
```

In [ ]:
```
from sys import argv, exit

import logging
logging.basicConfig(
     level = logging.DEBUG,
     filename = "log.txt",
     filemode = "w",
     format = "%(filename)10s:%(lineno)4d:%(message)s"
)
log = logging.getLogger()

import ply.lex as lex
from ply.lex import TOKEN
```

```
In [ ]: tokens = [
            "ID",  # identificador
            # numerais
            "NUM_NOTACAO_CIENTIFICA",  # ponto flutuante em notaçao científica
            "NUM_PONTO_FLUTUANTE",  # ponto flutuate
            "NUM_INTEIRO",  # inteiro
            # operadores binarios
            "ADICAO",  # +
            "SUBTRACAO",  # -
            "MULTIPLICACAO",  # *
            "DIVISAO",  # /
            "E_LOGICO",  # &&
            "OU_LOGICO",  # ||
            "DIFERENCA",  # <>
            "MENOR_IGUAL",  # <=
            "MAIOR_IGUAL",  # >=
            "MENOR",  # <
            "MAIOR",  # >
            "IGUALDADE",  # =
            # operadores unarios
            "NEGACAO",  # !
            # simbolos
            "ABRE_PAR",  # (
            "FECHA_PAR",  # )
            "ABRE_COL",  # [
            "FECHA_COL",  # ]
            "VIRGULA",  # ,
            "DOIS_PONTOS",  # :
            "ATRIBUICAO",  # :=
            # 'COMENTARIO', # {***}
        ]

In [ ]: reserved_words = {
            "se": "SE",
            "então": "ENTAO",
            "senão": "SENAO",
            "fim": "FIM",
            "repita": "REPITA",
            "flutuante": "FLUTUANTE",
            "retorna": "RETORNA",
            "até": "ATE",
            "leia": "LEIA",
            "escreva": "ESCREVA",
            "inteiro": "INTEIRO",
        }

        tokens = tokens + list(reserved_words.values())

In [ ]: digito = r"([0-9])"
        letra = r"([a-zA-ZáÁãÃàÀéÉíÍóÓõÕ])"
        sinal = r"([\-\+]?)"

        """
            id deve começar com uma letra
        """
```

```
        id = (
            r"(" + letra + r"(" + digito + r"+|_|" + letra + r")*)"
        )  # o mesmo que '((letra)(letra|_|([0-9]))*)'

        inteiro = r"(" + sinal + digito + r"+)"

        flutuante = (
            # r"(" + digito + r"+\." + digito + r"+?)"
            # (([-\+]?)([0-9]+)\.([0-9]+))'
            r'\d+[eE][-+]?\d+|(\.\d+|\d+\.\d*)([eE][-+]?\d+)?'
            # r'[-+]?[0-9]+(\.([0-9]+)?)'
            #r'[+-]?(\d+(\.\d*)?|\.\d+)([eE][+-]?\d+)?'
            #r"(([-\+]?)([0-9]+)\.([0-9]+))"
            )

        notacao_cientifica = (
            r"(" + sinal + r"([1-9])\." + digito + r"+[eE]" + sinal + digito + r"+)"
        )  # o mesmo que '(([-\+]?)([1-9])\.([0-9])+[eE]([-\+]?)([0-9]+))'
```

In [ ]:
```
        # Expressões Regulaes para tokens simples.
        # Símbolos.
        t_ADICAO    = r'\+'
        t_SUBTRACAO  = r'-'
        t_MULTIPLICACAO   = r'\*'
        t_DIVISAO = r'/'
        t_ABRE_PAR  = r'\('
        t_FECHA_PAR  = r'\)'
        t_ABRE_COL = r'\['
        t_FECHA_COL = r'\]'
        t_VIRGULA = r','
        t_ATRIBUICAO = r':='
        t_DOIS_PONTOS = r':'

        # Operadores Lógicos.
        t_E_LOGICO = r'&&'
        t_OU_LOGICO = r'\|\|'
        t_NEGACAO = r'!'

        # Operadores Relacionais.
        t_DIFERENCA = r'<>'
        t_MENOR_IGUAL = r'<='
        t_MAIOR_IGUAL = r'>='
        t_MENOR = r'<'
        t_MAIOR = r'>'
        t_IGUALDADE = r'='
```

In [ ]:
```
        @TOKEN(id)
        def t_ID(token):
            token.type = reserved_words.get(
                token.value, "ID"
            )  # não é necessário fazer regras/regex para cada palavra reservada
            # se o token não for uma palavra reservada automaticamente é um id
            # As palavras reservadas têm precedências sobre os ids
```

```
            return token

        @TOKEN(notacao_cientifica)
        def t_NUM_NOTACAO_CIENTIFICA(token):
            return token

        @TOKEN(flutuante)
        def t_NUM_PONTO_FLUTUANTE(token):
            return token

        @TOKEN(inteiro)
        def t_NUM_INTEIRO(token):
            return token
```

In [ ]:
```
t_ignore = " \t"

# t_COMENTARIO = r'(\{((.|\n)*?)\})'
# para poder contar as quebras de linha dentro dos comentarios
def t_COMENTARIO(token):
    r"(\{((.|\n)*?)\})"
    token.lexer.lineno += token.value.count("\n")
    # return token

def t_newline(token):
    r"\n+"
    token.lexer.lineno += len(token.value)

def define_column(input, lexpos):
    begin_line = input.rfind("\n", 0, lexpos) + 1
    return (lexpos - begin_line) + 1
```

In [ ]:
```
def t_error(token):

    # file = token.lexer.filename
    line = token.lineno
    # column = define_column(token.lexer.backup_data, token.lexpos)
    message = "Caracter ilegal '%s'" % token.value[0]

    # print(f"[{file}]:[{line},{column}]: {message}.")
    print(message)

    token.lexer.skip(1)

    # token.lexer.has_error = Trueb
```

In [ ]:
```
def main():
    # argv[1] = 'teste.tpp'
    aux = argv[1].split('.')
    if aux[-1] != 'tpp':
      raise IOError("Not a .tpp file!")
    data = open(argv[1])

    source_file = data.read()
    lexer.input(source_file)
```

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

In [ ]:
```
# Build the lexer.
__file__ = "01-compiladores-analise-lexica-tpplex.ipynb"
lexer = lex.lex(optimize=True,debug=True,debuglog=log)

if __name__ == "__main__":
    main()
```

```
---------------------------------------------------------------------------

OSError                                   Traceback (most recent call last)

01-compiladores-analise-lexica-tpplex.ipynb in <module>()
  4
  5 if __name__ == "__main__":
----> 6     main()


01-compiladores-analise-lexica-tpplex.ipynb in main()
  3     aux = argv[1].split('.')
  4     if aux[-1] != 'tpp':
----> 5       raise IOError("Not a .tpp file!")
  6     data = open(argv[1])
  7


OSError: Not a .tpp file!
```

In [ ]:
```
%%writefile teste.tpp

inteiro: a[10]
flutuante: b

inteiro func1(inteiro:x, flutuante:y)
  inteiro: res
  se (x > y) então
    res := x + y
  senão
    res := x * y
  fim
  retorna(res)
fim

func2(inteiro:z, flutuante:w)
  a := z
```

```
  b := w
fim

inteiro principal()
  inteiro: x,y
  flutuante: w
  a := 10 + 2
  leia(x)
  leia(w)
  w := .6 + 1.
  func2(1, 2.5)
  b := func1(x,w)
  escreva(b)
  retorna(0)
fim
```

```
Writing teste.tpp
```

In [ ]:
```
!python tpplex.py teste.tpp
```

```
python: can't open file 'tpplex.py': [Errno 2] No such file or directory
```