# -\*- coding: utf-8 -\*-

# Implementación de un scanner mediante la codificación de un Autómata

# Finito Determinista como una Matríz de Transiciones

# Autor: Dr. Santiago Conant, Agosto 2014 (modificado en Agosto 2015)

# A01064725

# A00816446

import sys

# tokens

INT = 100 # Número entero

FLT = 101 # Número de punto flotante

OPB = 102 # Operador binario

LRP = 103 # Delimitador: paréntesis izquierdo

RRP = 104 # Delimitador: paréntesis derecho

END = 105 # Fin de la entrada

CMA = 106 # Separador: comma

ASG = 107 # Asignación

IDT = 108 # Identificador

ERR = 200 # Error léxico: palabra desconocida

# Matriz de transiciones: codificación del AFD

# [renglón, columna] = [estado no final, transición]

# Estados > 99 son finales (ACEPTORES)

# Caso especial: Estado 200 = ERROR

# dig op ( ) raro esp . $ , = lcc \_

MT = [[ 1, OPB, LRP, RRP, 4, 0, 4, END, CMA, ASG, 5, 4], # edo 0 - estado inicial

[ 1, INT, INT, INT, INT, INT, 2, INT, INT, INT, 4, 4], # edo 1 - dígitos enteros

[ 3, ERR, ERR, ERR, 4, ERR, 4, ERR, ERR, ERR, 4, 4], # edo 2 - primer decimal flotante

[ 3, FLT, FLT, FLT, FLT, FLT, 4, FLT, FLT, FLT, 4, 4], # edo 3 - decimales restantes flotante

[ERR, ERR, ERR, ERR, 4, ERR, 4, ERR, ERR, ERR, 4, 4], # edo 4 - estado de error

[ 5, IDT, IDT, IDT, 4, IDT, 4, IDT, IDT, IDT, 5, 5]] # edo 5 - identificadores

# Filtro de caracteres: regresa el número de columna de la matriz de transiciones

# de acuerdo al caracter dado

def filtro(c):

"""Regresa el número de columna asociado al tipo de caracter dado(c)"""

if c == '0' or c == '1' or c == '2' or \

c == '3' or c == '4' or c == '5' or \

c == '6' or c == '7' or c == '8' or c == '9': # dígitos

return 0

elif c == '+' or c == '-' or c == '\*' or \

c == '/': # operadores

return 1

elif c == '(': # delimitador (

return 2

elif c == ')': # delimitador )

return 3

elif c == ' ' or ord(c) == 9 or ord(c) == 10 or ord(c) == 13: # blancos

return 5

elif c == '.': # punto

return 6

elif c == '$': # fin de entrada

return 7

elif c == ',': # comma

return 8

elif c == '=': # asignación

return 9

elif c.isalpha() and c.islower(): # letra en minúscula

return 10

elif c == '\_': # guíon bajo

return 11

else: # caracter raro

return 4

\_c = None # siguiente caracter

\_leer = True # indica si se requiere leer un caracter de la entrada estándar

# Función principal: implementa el análisis léxico

def obten\_token():

"""Implementa un analizador léxico: lee los caracteres de la entrada estándar"""

global \_c, \_leer

edo = 0 # número de estado en el autómata

lexema = "" # palabra que genera el token

while (True):

while edo < 100: # mientras el estado no sea ACEPTOR ni ERROR

if \_leer: \_c = sys.stdin.read(1)

else: \_leer = True

edo = MT[edo][filtro(\_c)]

if edo < 100 and edo != 0: lexema += \_c

if edo == INT:

\_leer = False # ya se leyó el siguiente caracter

print("Entero", lexema)

return INT

elif edo == FLT:

\_leer = False # ya se leyó el siguiente caracter

print("Flotante", lexema)

return FLT

elif edo == OPB:

lexema += \_c # el último caracter forma el lexema

print("Operador", lexema)

return OPB

elif edo == LRP:

lexema += \_c # el último caracter forma el lexema

print("Delimitador", lexema)

return LRP

elif edo == RRP:

lexema += \_c # el último caracter forma el lexema

print("Delimitador", lexema)

return RRP

elif edo == END:

print("Fin de expresion")

return END

elif edo == CMA:

lexema += \_c # el último caracter forma el lexema

print("Separador", lexema)

return CMA

elif edo == ASG:

lexema += \_c # el último caracter forma el lexema

print("Asignación", lexema)

return ASG

elif edo == IDT:

\_leer = False # ya se leyó el siguiente caracter

print("Identificador", lexema)

return IDT

else:

\_leer = False # el último caracter no es raro

print("ERROR! palabra ilegal", lexema)

return ERR