Práctica 07

DOCENTE	CARRERA	CURSO
MSc. Vicente Enrique Machaca Arceda	Escuela Profesional de Ingeniería de Software	Compiladores

PRÁCTICA	TEMA	DURACIÓN
07	Expresiones Regulares en Python	3 horas

1. Datos de los estudiantes

- Grupo: 1
- Git Hub: `https://github.com/CrazyDani17/practica7_compiladores`
- Cuaderno de códigos:
- Integrantes:
  - Guillermo Alemán
  - Marvik Del Carpio
  - Daniel Mendiguri
  - Daniela Vilchez

2. Competencias del curso

- Conocer las bases de la teoría de la computación para la implementación de un compilador.
- Implementar un compilador para un lenguaje de baja complejidad.

3. Competencias de la práctica

- Entender como convertir un autómata finito no determinista a un autómata finito determinista.

4. Equipos y materiales

- Latex
- Conección a internet
- Python
- Ply

5. Entregables

- Se debe elaborar un informe en **Latex** donde se responda a cada ejercicio de la Sección 7.
- Por cada 5 minutos de retraso, el alumno tendrá un punto menos.

6. Marco teórico

La implementación de un analizador léxico se puede realizar de varias formas. Todas ellas dependen de como definimos las expresiones regulares para reconocer cada token en nuestro código fuente. Podemos usar la librería re de Python para reconocer cada expresión o también podríamos utilizar la librería Ply, la cual es una forma avanzada y facil de implementar un analizador léxico.

A continuación mostraremos un ejemplo de como utilizar Ply para obtener un analizador léxico para una calculadora. Para más información puede revisar la documentación

```
# ------------------------------------------------------------
# tokenizer for a simple expression evaluator for
# numbers and +,-,*,/
# ------------------------------------------------------------
import ply.lex as lex

# List of token names. This is always required
tokens = ('NUMBER', 'PLUS', 'MINUS', 'TIMES', 'DIVIDE', 'LPAREN', 'RPAREN')
# Regular expression rules for simple tokens

t_PLUS = r'\+'
t_MINUS = r'-'
t_TIMES = r'\*'
t_DIVIDE = r'/'
t_LPAREN = r'\('
t_RPAREN = r'\)'

# A regular expression rule with some action code
def t_NUMBER(t):
  r'\d+'
  t.value = int(t.value) # guardamos el valor del lexema
  return t

# Define a rule so we can track line numbers
def t_newline(t):
  r'\n+'
  t.lexer.lineno += len(t.value)
  # A string containing ignored characters (spaces and tabs)
t_ignore = '\t'

# Error handling rule
def t_error(t):
  print("Illegal character  % s " % t.value[0])
  t.lexer.skip(1)

# Build the lexer
lexer = lex.lex()

# Test it out
```

```
data = '''3 + 4 * 10 +
-20 *2'''

# Give the lexer some input
lexer.input(data)

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

7. Ejercicios

1. Implemente el analizador léxico para el lenguaje propuesto. Su programa deberá leer el código fuente de archivo en disco (debe proporcionar varios ejemplos) y luego deberá mostrar todos los tokens de manera similar al ejemplo mostrado en la Sección 5

```
# ------------------------------------------------------------
# tokenizer for a simple expression evaluator for
# numbers and +,-,*,/
# ------------------------------------------------------------
import ply.lex as lex

# List of token names. This is always required
tokens =
    ('mision','si','sino','pinocho','PINOCHO','numero','NUMERO','decimal','DECIMAL',
'mostrar','texto','TEXTO','descansito','yoyo','devuelve','chacha','CHACHA','mas','menos',
'por','y','o','dividir','mayor','menor','pizquierdo','pderecho','cizquierdo','cderecho',
'coma','igual','menor_igual','mayor_igual','diferente','identificador','lizquierdo',
'lderecho','division','verdad','mentira')

# Regular expression rules for simple tokens
t_mision = r'mision'
t_si = r'si'
t_sino = r'sino'
t_pinocho = r'pinocho'
t_numero = r'numero'
t_decimal = r'decimal'
t_mostrar = r'mostrar'
t_texto = r'texto'
t_descansito = r'descansito'
t_yoyo = r'yoyo'
t_devuelve = r'devuelve'
t_chacha = r'chacha'

#operadores
t_mas = r'\+'
t_menos = r'\-'
t_por = r'\*'
t_division = r'\/'
```

```
#operadores logicos
t_y = r'\#y'
t_o = r'\#o'

#Mayor y menor
t_mayor = r'\>'
t_menor = r'\<'

#Parentesis
t_pizquierdo = r'\('
t_pderecho = r'\)'

#Llaves
t_lizquierdo = r'\{'
t_lderecho = r'\}'

#Corchetes
t_cizquierdo = r'\['
t_cderecho = r'\]'

#Otros
t_coma = r'\,'
t_igual = r'\='
t_menor_igual = r'\<\='
t_mayor_igual = r'\>\='
t_diferente = r'\<\>'

def t_identificador(t):
  r'(?!decimal|si|sino|pinocho|numero|mostrar|texto|descansito|yoyo|devuelve|chacha|
  mision|mentira|verdad)[a-zA-Z]+[\w]*'
  try:
    t.value = t.value
  except ValueError:
    t.value = 0
  return t

# A regular expression rule with some action code
def t_mentira(t):
  r'mentira'
  t.value = False # guardamos el valor del lexema
  return t

def t_verdad(t):
  r'verdad'
  t.value = True # guardamos el valor del lexema
  return t

def t_NUMERO(t):
  r'\d+(?!\.)'
  try:
    t.value = int(t.value) # guardamos el valor del lexema
  except ValueError:
    t.value = 0
  return t
```

```
def t_DECIMAL(t):
  r'(0|[1-9][0-9]*)\.[0-9]*'
  try:
    t.value = float(t.value)
  except ValueError:
    t.value = 0
  return t

def t_CHACHA(t):
  r'\'(\W|\w)\''
  t.value = t.value # guardamos el valor del lexema
  return t

def t_TEXTO(t):
  r'\"(\W|\w)+\"'
  t.value = t.value # guardamos el valor del lexema
  return t

# Define a rule so we can track line numbers
def t_newline(t):
  r'\n+'
  t.lexer.lineno += len(t.value)
# A string containing ignored characters (spaces and tabs)
t_ignore = '\t| '
# Error handling rule
def t_error(t):
  print("Illegal character '%s'" % t.value[0])
  t.lexer.skip(1)
# Build the lexer
lexer = lex.lex()
# Test it out
data = "mision llama(){decimal x = 38888 + (4*5) - 2.5}"
# Give the lexer some input
lexer.input(data)

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

8. Rúbricas

Rúbrica	Cumple	Cumple con obs.	No cumple
Informe: El informe debe estar en Latex, con un formato limpio (buena presentación) y facil de leer.	3	1.5	0
textbfExpresiones regulares: Define todas las expresiones regulares de su lenguaje.	4	2	0
Implementación: Implementa el analizador l exico. Si el alumno NO utiliza Ply (o alguna similar) tendra mayor nota.	10	5	0
Presentación: El alumno demuestra dominio del tema y conoce con exactitud cada parte de su trabajo.	3	1.5	0
Errores ortográficos: Por cada error ortográfico, se le descontará un punto.	-	-	-