 **Universidad de Guadalajara**

Centro Universitario de Ciencias Exactas E Ingenierías

División de Electrónica y Computación

Ingeniería en computación

**D02 SEMINARIO DE SOLUCION DE PROBLEMAS DE TRADUCTORES DE LENGUAJES II**

**Mini generador léxico**

**216788333 Alejandro de Jesús Romo Rosales**

**10/09/2020**

**Objetivo**

Genera un pequeño analizador léxico, que identifique los siguientes tokens (identificadores y números reales) construidos de la siguiente manera.

identificadores = letra(letra|digito)\*

Real = entero.entero+

**Introducción**

Hacer un analizador léxico significa leer los caracteres de entrada y elaborar como salida una secuencia de componentes léxicos que utiliza el analizador sintáctico para hacer el análisis.

El analizador léxico es la primera fase de un compilador.

Su principal función consiste en leer los caracteres de entrada y elaborar como salida una secuencia de componentes léxicos que utiliza el analizador sintáctico para hacer el análisis. Esta interacción, suele aplicarse convirtiendo al analizador léxico en una subrutina o corrutina del analizador sintáctico. Recibida la orden "obtén el siguiente componente léxico" del analizador sintáctico, el analizador léxico lee los caracteres de entrada hasta que pueda identificar el siguiente componente léxico.

**Metodología**

Posterior a una investigación sobre el tema en cuestión, implementar seleccionar un lenguaje que soporte el paradigma orientado a objetos para así poder modelar de una manera más efectiva y eficiente la abstracción de las clases, por ejemplo, la clase Token. Una vez realizado esto se procederá a programar el analizador léxico mediante el uso de funciones y finalizando esto predefinir una serie de pruebas las cuales nos ayudaran a corroborar el su debido comportamiento.

**Materiales**

1. Python 3.7
2. VS Code
3. Windows 10

**Desarrollo**

Lo primero que haremos será definir la clase Tokens.

"""""""""""""""""""""""""""""

"" TOKENS

"""""""""""""""""""""""""""""

# TipoToken

TT\_INT        = "INT"

TT\_FLOAT      = "FLOAT"

TT\_EQ         = "IGUAL"

TT\_IDENTIFIER = "IDENTIFICADOR"

TT\_ASIGN      = "ASIGNACION"

DIGIT = '0123456789'

LETTERS = 'abcdefghijklmnopqrstwxyzABCDEFGHIJKLMNOPQRSTWXYZ'

class Token:

    def \_\_init\_\_(self, tpe, value=None):

        self.\_type = tpe

        self.\_value = value

    def \_\_repr\_\_(self):

        if self.\_value: return f'{self.\_type}:{self.\_value}'

        return f'{self.\_type}'

    # Regresa el valor del Token

    def Value(self):

        if self.\_value != None:

            return self.\_value

    # Regresa el tipo del Token

    def Type(self):

        return self.\_type

""""""""""""""""""""""""""""""

Esto nos permitirá definir los componentes de nuestro texto de entrada.

Posteriormente definiremos los tipos de errores que podremos encontrar así como la posición ya que un buen compilador siempre le hace mención al programador donde esta el error.

"""""""""""""""""""""""""""""

"" ERRORES

"""""""""""""""""""""""""""""

class Error:

    def \_\_init\_\_(self, pos\_start, pos\_end, error\_name, details):

        self.\_pos\_start = pos\_start

        self.\_pos\_end = pos\_end

        self.\_error\_name = error\_name

        self.\_details = details

    def as\_string(self):

        result  = f'{self.\_error\_name}: {self.\_details}\n'

        pos = self.\_pos\_start.\_ln+1

        result += f'Archivo {self.\_pos\_start.\_fn}, line {pos}'

        return result

class IllegalCharError(Error):

    def \_\_init\_\_(self, pos\_start, pos\_end, details):

        super().\_\_init\_\_(pos\_start, pos\_end, 'Caracter Invalido', details)

# Error en la definicion de la variable

class IllegalVariableError(Error):

    def \_\_init\_\_(self, pos\_start, pos\_end, details):

        super().\_\_init\_\_(pos\_start, pos\_end, 'Declaración de variable invalida ', details)

class Position:

    def \_\_init\_\_(self, idx, ln, col, fn, fntxt):

        self.\_idx = idx

        self.\_ln = ln

        self.\_col = col

        self.\_fn = fn

        self.\_fntxt = fntxt

    def advance(self, current\_char):

        self.\_idx += 1

        self.\_col += 1

        if current\_char == '\n':

            self.\_ln += 1

            self.\_col += 0

        return self

    def copy(self):

        return Position(self.\_idx, self.\_ln, self.\_col, self.\_fn, self.\_fntxt)

""""""""""""""""""""""""""""""

Con esto hecho, definiremos la lógica de nuestro analizador, lo cual será analizar el texto de entrada para obtener tokens, o bien, errores.

"""""""""""""""""""""""""""""

"" Mini analizador léxico

"""""""""""""""""""""""""""""

class Lexer:

    def \_\_init\_\_(self, fn, text):

        self.\_fn = fn

        self.\_text = text

        self.\_pos = Position(-1, 0, -1, fn, text)

        self.\_current\_char = None

        self.advance()

    def advance(self):

        self.\_pos.advance(self.\_current\_char)

        self.\_current\_char = self.\_text[self.\_pos.\_idx] if self.\_pos.\_idx < len(self.\_text) else None

    def make\_number(self):

        num\_str = ''

        dot\_count = 0

        while self.\_current\_char != None and self.\_current\_char in DIGIT + '.':

            if self.\_current\_char == '.':

                if dot\_count == 1: break

                dot\_count += 1

                num\_str += '.'

            else:

                num\_str += self.\_current\_char

            self.advance()

        if dot\_count == 0:

            return Token(TT\_INT, int(num\_str))

        else:

            return Token(TT\_FLOAT, float(num\_str))

    def make\_identifier(self):

        id\_str = ''

        pos = self.\_pos.copy()

        while self.\_current\_char != None and self.\_current\_char in LETTERS:

            id\_str += self.\_current\_char

            self.advance()

        return Token(TT\_IDENTIFIER, id\_str)

    def analyze(self):

        tokens = []

        count = self.\_text.count('=')

        if count > 1:

            pos = self.\_pos.copy()

            char = self.\_current\_char

            self.advance()

            return [], IllegalVariableError(pos, self.\_pos, '-> ', self.\_text)

        elif count == 1:

            name = self.\_text.split('=')[0]

            tokens.append(Token(TT\_IDENTIFIER, name))

            for n in range(len(name)):

               self.advance()

            tokens.append(self.make\_tokens(tokens)[0])

            return tokens, None

        else:

            tokens = self.make\_tokens(tokens)[0]

            return tokens, None

    def make\_tokens(self, tokens):

        # tokens = []

        while self.\_current\_char != None:

            if self.\_current\_char in '\t' or self.\_current\_char == ' ':

                self.advance()

            elif self.\_current\_char in DIGIT:

                tokens.append(self.make\_number())

            elif self.\_current\_char == '=':

                tokens.append(Token(TT\_ASIGN))

                self.advance()

            else:

                pos = self.\_pos.copy()

                char = self.\_current\_char

                self.advance()

                return [], IllegalCharError(pos, self.\_pos, "'" + char + "'")

        return tokens, None

""""""""""""""""""""""""""

def make\_identifier(text):

    num = float(' '.join(map(str, text)).split('=')[-1])

    return num

def run(fn, text):

    lexer = Lexer(fn, text)

    tokens, error = lexer.analyze()

    return tokens, error

Al finalizar esto, tendremos completo nuestro mini analizador léxico, por lo cual sigue hacer la entrada del texto así como lo que esto conlleva.

import lexico

def isExist(text):

    try:

        globals()[text]

        return True

    except:

        return False

if \_\_name\_\_ == '\_\_main\_\_':

    while True:

        text = input('shell > ')

        tokens, error = lexico.run('<stdin>', text)

        # Comando para salir de la consola

        if text == '\c':

            break

        # Si el texto ingresado es una variable previamente guarda se manda a llamar

        elif isExist(text):

            print("\t", globals()[text])

        else:

            # Imprime el error en caso de que haya encontrado uno

            if error:

                print(error.ToString())

            else:

                expression = lexico.make\_identifier(text)

                if expression != None:

                    # De haber hecho la declaracion de una variable se añade al entorno junto con su valor

                    globals()[(tokens[0].Value())] = expression

                # Imprime los tokens encontrados con sus respectivos valores

                print("\tTOKENS->", tokens)

**Resultados**

Con el analizador léxico programado se procedieron a realizar diversas pruebas para la verificación del funcionamiento del mismo.



 