



Tecnológico Nacional de México

Instituto Tecnológico de Pachuca

Tema:

Implementación del analizador sintáctico al léxico 5.2

Ingeniería en Sistemas Computacionales

7to Semestre Grupo: B

Materia: Lenguajes y Autómatas I

Profesor: Baume Lazcano Rodolfo

Equipo:

Martínez González José Pablo

Morales Ordoñez Yesenia

Lara Lopez Marco Antonio

Muñoz Castillo Ariana

SEMESTRE: Febrero – Junio 2024

31 – Mayo – 2024

TABLA DE TOKENS

Una tabla de tokens en un analizador léxico es una estructura de datos utilizada para almacenar información sobre los tokens identificados durante el proceso de análisis léxico de un programa o texto. Un token es una secuencia de caracteres con un significado coherente en el lenguaje de programación o en el contexto de análisis.

TOKEN	TIPO DE TOKEN	Expresion regular
Suma	Palabras clave	<code>r'\+'</code>
Resta	Palabras clave	<code>r'\-'</code>
Multi	Palabras clave	<code>r'*'</code>
Divi	Palabras clave	<code>r'\/'</code>
Igual	Palabras clave	<code>r'\='</code>
Numero	Palabras clave	<code>r'\d+'</code>
Ignorar	Palabras clave	<code>r'\t'</code>
Nueva Linea	Palabras clave	<code>r'\n+'</code>
Error	Palabras clave	<code>r"Carácter no válido: '%s'"</code>
Par	Palabra clave	<code>r'\('</code>
Corchete	Palabra clave	<code>r'\['</code>
Llave	Palabra clave	<code>r'\{'</code>

```
3
4 # Definición de tokens
5 tokens = ['Numero', 'Suma', 'Resta', 'Multi', 'Divi', 'Igual', 'Par', 'Corchete', 'Llave']
6
```

REGLAS DE COINCIDENCIA

Son instrucciones definidas por el programador en un analizador léxico para identificar y clasificar los tokens en un flujo de caracteres.

Estas reglas se utilizan para dividir el código fuente en unidades léxicas significativas, como palabras clave, identificadores, operadores, números, etc.

`"r'\d+', r'\-', r'*', r'\/', r'\=', r'\d+', r'\t', r'\n+', r"Carácter no válido: '%s'"`

```
7 # Expresiones regulares para tokens
8 t_Suma = r'\+'
9 t_Resta = r'\-'
10 t_Multi = r'\*'
11 t_Divi = r'\/'
12 t_Igual = r'\='
13 t_Par = r'\('
14 t_Corchete = r'\['
15 t_Llave = r'\{'
```

ASOCIAR REGLAS

Implica definir qué hacer cuando se encuentra una coincidencia entre el patrón de una regla y el texto de entrada. Estas acciones pueden incluir la creación de un token, el registro de información adicional sobre el token, la ejecución de algún código específico o simplemente la omisión de ciertos caracteres.

```
# Función personalizada creada por el usuario para imprimir los tokens
def imprimir_token(token):
    print(f"Mi Token (tipo = {token.type}, valor = {token.value}, Línea Del Token = {token.lineno}, Posi

# Aquí se imprimen los tokens que reconoce nuestro analizador
while True:
    token = lexer.token()
    if not token:
        break
    imprimir_token(token)
```

MANEJO DE CASOS ESPECIALES

El manejo de casos especiales en un analizador léxico es crucial para garantizar que el análisis del código fuente se realice correctamente, incluso en situaciones que pueden desviarse de las reglas estándar.

```
# En este apartado es en donde nosotros manejamos los errores
def t_error(t):
    print("Carácter no válido: '%s'" % t.value[0])
    t.lexer.skip(1)
```

INTRODUCCIÓN

El lenguaje de programación utilizado fue realizado en Python su función es procesar componentes básicos de una expresión matemática. El analizador utiliza la biblioteca “Ply” para la construcción de compiladores en este caso utilizando el **lex** para el léxico, y el **yacc** para el sintáctico.

DESCRIPCIÓN DEL LENGUAJE

El subconjunto de lenguaje analizado se compone de expresiones aritméticas básicas que incluyen números enteros, los operadores suma (+), resta (-), multiplicación (*), división (/). Las expresiones se pueden formar utilizando paréntesis para agrupar sub expresiones.

PROPÓSITO DEL ANALIZADOR SINTÁCTICO

Es ser el encargado de verificar la gramática utilizada en las entradas para que esta sea correcta de acuerdo a las reglas ya establecidas en nuestro lenguaje de programación para ello se tomaría en cuenta la serie de tokens que generamos en lo léxico ya que de no ser así se arrojaría un mensaje de error en la sintaxis.

SÍMBOLOS TERMINALES

- | | |
|-------------------------|--|
| 1. Numero | Solo entero |
| 2. Suma | Símbolo característico de suma + |
| 3. Resta | Símbolo característico de resta - |
| 4. Multi | Símbolo característico de multiplicación * |
| 5. Divi | Símbolo característico de división / |
| 6. Delimitadores | Son (, [, { |

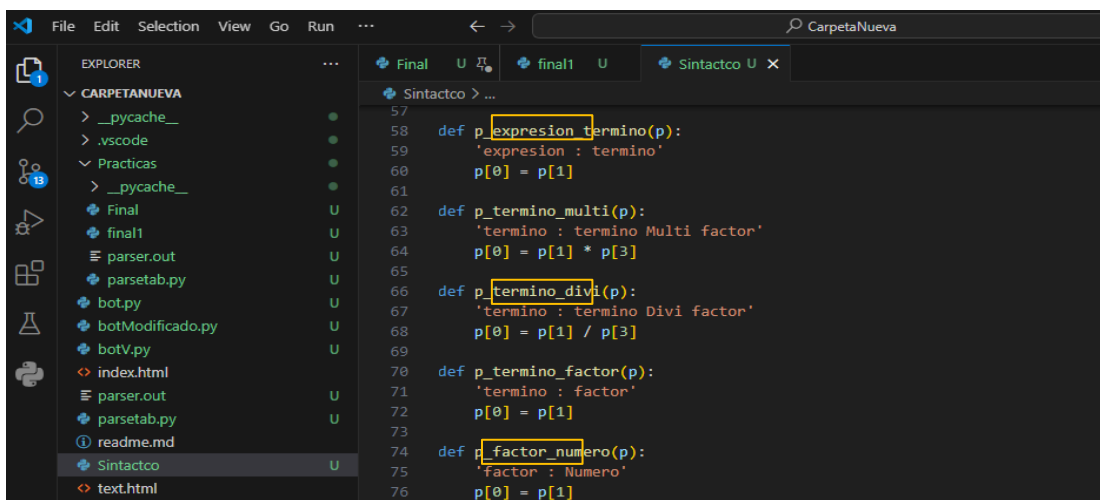
```

7 # Expresiones regulares para tokens
8 t_Suma = r'\+'
9 t_Resta = r'\-'
10 t_Multi = r'\*'
11 t_Divi = r'\/'
12 t_Igual = r'\='
13 t_Par = r'\('
14 t_Corchete = r'\['
15 t_Llave = r'\{'

```

SÍMBOLOS NO TERMINALES

- **expresion:** representa el total de entrada (toda la operación a evaluar)
- **termino:** representa una parte esencial de la operación (valor numérico)
- **factor:** representa el complemento requerido para ser evaluado (signo o determinantes)



DETERMINANTES

Los declarados como determinantes dentro de nuestro código son:

- Paréntesis sencillo abierto '('
- Corchete sencillo abierto '['
- Llave sencilla abierto '{'

PRODUCCIÓN Y REGLAS

Dentro de estas producciones y reglas determinamos de qué forma deberían establecerse el orden de las entras ya que si este orden no se lleva a cabo podría recaer a un error y estos errores se inclinan a lo que es la procedencia y la asociatividad ya que son punto

clave que como analizador sintáctico identifica.

- **expresion** : termino Suma termino
- **expresion** : termino Resta termino
- **expresion**: conjunto de terminos
- **termino** : termino Multi factor
- **termino** : termino Divi factor
- **termino** : factor
- **factor** : Número

SÍMBOLOS INICIALES

El símbolo inicial de la gramática es 'expresión'

PROCEDENCIA Y ASOCIATIVIDAD

Los operadores determinan el orden en que se evalúan los operadores en una expresión. Los operadores con mayor precedencia se evalúan antes que los operadores con menor precedencia.

Multiplicación y división tienen una mayor precedencia que los operadores aditivos (suma y resta).

Precedencia

Operador de multiplicación (*) tiene precedencia alta

Operador de división (/) tiene precedencia alta

Operador de suma (+) tiene precedencia baja

Operador de resta (-) tiene precedencia baja

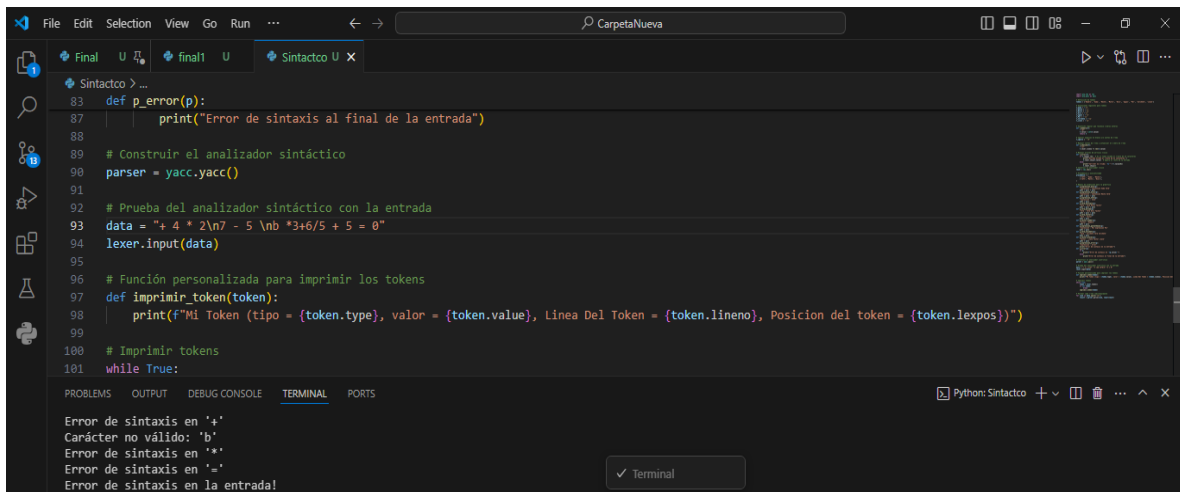
Asociatividad

Operador de multiplicación (*) asociativo por la izquierda

Operador de división (/) asociativo por la izquierda

Operador de suma (+) asociativo por la izquierda

Operador de resta (-) asociativo por la izquierda



```
File Edit Selection View Go Run ... CarpetaNueva
Sintactico.py
83 def p_error(p):
84     print("Error de sintaxis al final de la entrada")
85
86 # Construir el analizador sintáctico
87 parser = yacc.yacc()
88
89 # Prueba del analizador sintáctico con la entrada
90 data = "+ 4 * 2\n7 - 5 \nb *3+6/5 + 5 = 0"
91 lexer.input(data)
92
93 # Función personalizada para imprimir los tokens
94 def imprimir_token(token):
95     print(f"Mi Token (tipo = {token.type}, valor = {token.value}, Línea Del Token = {token.lineno}, Posición del token = {token.lexpos})")
96
97 # Imprimir tokens
98 while True:
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python: Sintactico

```
Error de sintaxis en '+'
Carácter no válido: 'b'
Error de sintaxis en '*'
Error de sintaxis en '='
Error de sintaxis en la entrada!
```

COMENTARIOS Y ANOTACIONES

1. # Definición de tokens
2. # Expresiones regulares para tokens
3. # Expresión regular que reconoce números enteros
4. # Ignorar espacios en blanco y/o saltos de línea
5. # Manejar saltos de línea y actualizar el número de línea
6. # Manejar errores de análisis léxico
7. # Construir el analizador léxico
8. # Precedencia y asociatividad
9. # Reglas de producción para la gramática
10. # Construir el analizador sintáctico
11. # Prueba del analizador sintáctico con la entrada
12. # Función personalizada para imprimir los tokens
13. # Imprimir tokens
14. # Parsear cada línea individualmente

- Construcción de analizador sintáctico

```

80 def p_expression_error(p):
81     'expression : error'
82     print("Error de sintaxis en la entrada!")
83 def p_error(p):
84     if p:
85         print(f"Error de sintaxis en '{p.value}'")
86     else:
87         print("Error de sintaxis al final de la entrada")
88
89 # Construir el analizador sintáctico
90 parser = yacc.yacc()
91
92 # Prueba del analizador sintáctico con la entrada
93 data = "(3 + 4 * 2\n7 - 5 \nb *3+6/5 + 5 = 0]"
94 lexer.input(data)
95
96 # Función personalizada para imprimir los tokens
97 def imprimir_token(token):
98     print(f"Mi Token (tipo = {token.type}, valor = {token.value}, Línea Del Token = {token.lineno}, Posición del token = {token.lexpos})")
99
100 # Imprimir tokens
101 while True:
102     token = lexer.token()
103     if not token:
104         break
105     imprimir_token(token)
106
107 # Parsear cada línea individualmente
108 for line in data.split('\n'):
109     result = parser.parse(line, lexer=lexer)

```

- Ejecución correcta

```

d:\in\Desktop\6to Semestre\CarpetaNueva\Sintactico>
Mi Token (tipo = Numero, valor = 3, Línea Del Token = 1, Posición del token = 0)
Mi Token (tipo = Suma, valor = +, Línea Del Token = 1, Posición del token = 2)
Mi Token (tipo = Numero, valor = 4, Línea Del Token = 1, Posición del token = 4)
Mi Token (tipo = Multi, valor = *, Línea Del Token = 1, Posición del token = 6)
Mi Token (tipo = Numero, valor = 2, Línea Del Token = 1, Posición del token = 8)
Mi Token (tipo = Numero, valor = 7, Línea Del Token = 2, Posición del token = 10)
Mi Token (tipo = Resta, valor = -, Línea Del Token = 2, Posición del token = 12)
Mi Token (tipo = Numero, valor = 5, Línea Del Token = 2, Posición del token = 14)
Mi Token (tipo = Resta, valor = -, Línea Del Token = 2, Posición del token = 16)
Mi Token (tipo = Numero, valor = 5, Línea Del Token = 2, Posición del token = 18)
Mi Token (tipo = Suma, valor = +, Línea Del Token = 2, Posición del token = 20)
Mi Token (tipo = Numero, valor = 5, Línea Del Token = 2, Posición del token = 22)
Mi Token (tipo = Suma, valor = +, Línea Del Token = 2, Posición del token = 23)
Mi Token (tipo = Numero, valor = 0, Línea Del Token = 2, Posición del token = 26)
PS C:\Users\admin\Desktop\6to Semestre\CarpetaNueva>

```

- Detecta un carácter no valido en este caso texto


```
def p_factor_braces(p):
    'factor : Llave factor Llave'
    p[0] = p[2]

def p_expression_error(p):
    'expression : error'
    print("Error de sintaxis en la entrada!")

def p_error(p):
    if p:
        print(f"Error de sintaxis en '{p.value}'")
    else:
        print("Error de sintaxis al final de la entrada")

# Construir el analizador sintáctico
parser = yacc.yacc()

# Prueba del analizador sintáctico con la entrada
data = "3 + 4 * 2\n7 - 5 \nb 5 + 5 + 0"
lexer.input(data)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Python: Sintactico

Mi Token (tipo = Numero, valor = 7, Línea Del Token = 2, Posición del token = 18)
Mi Token (tipo = Resta, valor = -, Línea Del Token = 2, Posición del token = 12)
Mi Token (tipo = Numero, valor = 5, Línea Del Token = 2, Posición del token = 14)
Carácter no válido: 'b'
Mi Token (tipo = Numero, valor = 5, Línea Del Token = 3, Posición del token = 19)
Mi Token (tipo = Suma, valor = +, Línea Del Token = 3, Posición del token = 21)
Mi Token (tipo = Numero, valor = 5, Línea Del Token = 3, Posición del token = 23)
Mi Token (tipo = Suma, valor = +, Línea Del Token = 3, Posición del token = 25)
Mi Token (tipo = Numero, valor = 0, Línea Del Token = 3, Posición del token = 27)
Carácter no válido: 'b'
PS C:\Users\admin\Desktop\6to Semestre\CarpetaNueva>

- Detecta el error de ingresar más caracteres de lo determinado en este caso se había asignado un límite de 31 valores.

```
def p_error(p):
    if p:
        print(f"Error de sintaxis en '{p.value}'")
    else:
        print("Error de sintaxis al final de la entrada")

# Construir el analizador sintáctico
parser = yacc.yacc()

# Prueba del analizador sintáctico con la entrada
data = "(3 + 4 * 2\n7 - 5 \nb * 3 + 6 / 5 + 5 = 0)"
lexer.input(data)

# Función personalizada para imprimir los tokens
def imprimir_token(token):
    print(f"Mi Token (tipo = {token.type}, valor = {token.value}, Línea Del Token = {token.lineno}, Posición del token = {token.pos})")

# Imprimir tokens
for token in lexer.tokens:
    imprimir_token(token)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Python: Sintactico

Mi Token (tipo = Numero, valor = 2, Línea Del Token = 1, Posición del token = 9)
Mi Token (tipo = Numero, valor = 7, Línea Del Token = 2, Posición del token = 11)
Mi Token (tipo = Resta, valor = -, Línea Del Token = 2, Posición del token = 13)
Mi Token (tipo = Numero, valor = 5, Línea Del Token = 2, Posición del token = 15)
Carácter no válido: 'b'
Mi Token (tipo = Multi, valor = *, Línea Del Token = 3, Posición del token = 20)
Mi Token (tipo = Numero, valor = 3, Línea Del Token = 3, Posición del token = 21)
Mi Token (tipo = Suma, valor = +, Línea Del Token = 3, Posición del token = 22)
Mi Token (tipo = Numero, valor = 6, Línea Del Token = 3, Posición del token = 23)
Mi Token (tipo = Divi, valor = /, Línea Del Token = 3, Posición del token = 24)
Mi Token (tipo = Numero, valor = 5, Línea Del Token = 3, Posición del token = 25)
Mi Token (tipo = Suma, valor = +, Línea Del Token = 3, Posición del token = 27)
Mi Token (tipo = Numero, valor = 5, Línea Del Token = 3, Posición del token = 29)
Mi Token (tipo = Igual, valor = =, Línea Del Token = 3, Posición del token = 31)
Mi Token (tipo = Numero, valor = 0, Línea Del Token = 3, Posición del token = 33)
Entrada excede el límite de 31 caracteres.

- Detecta signos declarados como lo es el =, y determinates no especificados como lo son llave, parentesis y corchete en estado cerrado.

```

File Edit Selection View Go Run ...
CarpetaNueva

EXPLORER
  CARPETANUEVA
    _pycache_
    .vscode
    Practicas
      _pycache_
        Final
        final1
        parser.out
        parsetab.py
        bot.py
        botModificado.py
        botV.py
        index.html
        parser.out
        parsetab.py
        readme.md
        Sintactco
        text.html

101 # Construir el analizador sintáctico
102 parser = yacc.yacc()
103
104 # Prueba del analizador sintáctico con la entrada
105 data = "(3 + 4 * 2\n7 - 5 \nb 5 + 5 = 0)"
106 lexer.input(data)
107
108 # Función personalizada para imprimir los tokens
109 def imprimir_token(token):
110     print(f"Mi Token (tipo = {token.type}, valor = {token.value}, Línea Del Token = {token.lineno}, Posición del token = {token.pos})")
111
112 # Imprimir tokens
113 while True:
114     token = lexer.token()
115     if not token:
116         break
117     imprimir_token(token)
118
119 # Parsear cada línea individualmente
120 for line in data.split('\n'):
121     lexer.input(line)
122     token = lexer.token()
123     while token:
124         imprimir_token(token)
125         token = lexer.token()

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python: Sintactco
Mi Token (tipo = Suma, valor = +, Línea Del Token = 3, Posición del token = 22)
Mi Token (tipo = Numero, valor = 5, Línea Del Token = 3, Posición del token = 24)
Mi Token (tipo = Igual, valor = =, Línea Del Token = 3, Posición del token = 26)
Mi Token (tipo = Numero, valor = 0, Línea Del Token = 3, Posición del token = 28)
Carácter no válido: ')'
Error de sintaxis al final de la entrada
Carácter no válido: 'b'
Error de sintaxis en '='
Carácter no válido: ')'
Error de sintaxis en la entrada!
PS C:\Users\admin\Desktop\6to Semestre\CarpetaNueva>
Ln 105, Col 37, Spaces: 4, UTF-8, CRLF, Python, 3.12.3 64-bit (Microsoft Store)

```

CÓDIGO

```

import ply.lex as lex
import ply.yacc as yacc

# Definición de tokens
tokens = ['Numero', 'Suma', 'Resta', 'Multi', 'Divi', 'Igual', 'Par',
          'Corchete', 'Llave']

# Expresiones regulares para tokens
t_Suma = r'\+'
t_Resta = r'\-'
t_Multi = r'\*'
t_Divi = r'\/'
t_Igual = r'\='
t_Par = r'\('
t_Corchete = r'\['
t_Llave = r'\{'

# Expresión regular que reconoce números enteros
def t_Numero(t):

```

```

    r'\d+'
    t.value = int(t.value)
    return t

# Ignorar espacios en blanco y/o saltos de línea
t_ignore = ' \t'

# Manejar saltos de línea y actualizar el número de línea
def t_newline(t):
    r'\n+'
    t.lexer.lineno += len(t.value)

# Manejar errores de análisis léxico
def t_error(t):
    if t.lexpos > 31: # Si el token excede el límite de 15 caracteres
        print("Entrada excede el límite de 31 caracteres.")
        t.lexer.skip(t.value) # Ignora el resto de la entrada
    else:
        print("Carácter no válido: '%s'" % t.value[0])
        t.lexer.skip(1)

# Construir el analizador léxico
lexer = lex.lex()

# Precedencia y asociatividad
precedence = (
    ('left', 'Suma', 'Resta'),
    ('left', 'Multi', 'Divi'),
)

# Reglas de producción para la gramática
def p_expression_plus(p):
    'expression : expression Suma term'
    p[0] = p[1] + p[3]
def p_expression_minus(p):
    'expression : expression Resta term'
    p[0] = p[1] - p[3]
def p_expression_term(p):
    'expression : term'
    p[0] = p[1]
def p_term_multiply(p):
    'term : term Multi factor'
    p[0] = p[1] * p[3]
def p_term_divide(p):
    'term : term Divi factor'
    p[0] = p[1] / p[3]

```

```

def p_term_factor(p):
    'term : factor'
    p[0] = p[1]
def p_factor_number(p):
    'factor : Numero'
    p[0] = p[1]
def p_expression_parentheses(p):
    'expression : Par expression Par'
    p[0] = p[2]
def p_term_brackets(p):
    'term : Corchete term Corchete'
    p[0] = p[2]
def p_factor_braces(p):
    'factor : Llave factor Llave'
    p[0] = p[2]
def p_expression_error(p):
    'expression : error'
    print("Error de sintaxis en la entrada!")
def p_error(p):
    if p:
        print(f"Error de sintaxis en '{p.value}'")
    else:
        print("Error de sintaxis al final de la entrada")

# Construir el analizador sintáctico
parser = yacc.yacc()

# Prueba del analizador sintáctico con la entrada
data = "(3 + 4 * 2\n7 - 5 \nb *3+6/5 + 5 = 0]"
lexer.input(data)

# Función personalizada para imprimir los tokens
def imprimir_token(token):
    print(f"Mi Token (tipo = {token.type}, valor = {token.value}, Linea Del Token = {token.lineno}, Posicion del token = {token.lexpos})")

# Imprimir tokens
while True:
    token = lexer.token()
    if not token:
        break
    imprimir_token(token)

# Parsear cada línea individualmente
for line in data.split('\n'):

```

```
result = parser.parse(line, lexer=lexer)
```