

Tecnológico Nacional de México
Campus Pachuca



4.3 Proyecto Analizador Léxico

L e n g u a j e s A u t ó m a t a s

Catedrático: Rodolfo Baume Lazcano

Alumnos:

- | | | | |
|-------------------------------|----------------|--------------------------|----------------|
| • Sánchez Badillo Adrián | N.C.: 21200629 | • Morales Mateos Johanna | N.C.: 21200619 |
| • Taboada Hernandez Erwin O. | N.C.: 21200631 | | |
| • Garrido Virgen J. Alejandro | N.C.: 21200602 | | |
| • Serrano González Uriel A. | N.C.: 21200630 | | |

E q u i p o

Nuestro proyecto

Nuestro proyecto se centra en desarrollar un analizador léxico para el lenguaje de **álgebra básica**. El analizador léxico (parte fundamental de un compilador), ya que se encarga de analizar el texto de entrada y dividirlo en unidades más pequeñas llamadas **tokens**.

$$(ax)^b = a^b \bullet x^b$$

Tabla de tokens:

01

Se usara la siguiente tabla de tokens (fase preliminar, puede cambiar respecto al programa final):

Tokens	Alfabeto	Patrón
Operadores	"+", "-", "*", "/", "^", "<", ">", "="	'[+\\-*/=]'
Paréntesis	"(", ")", "[", "]"	'[()]'
Números	[0-9]	'\\b\\d+(\\.\\d+)?\\b'
Variables	[a-z, A-Z]	'\\b[a-z,A-Z]\\b'
Espacio	\\s+	(ignorar)

Token Tipo	Expresión Regular	Explicación
NUMBER	\\b\\d+(\\.\\d+)?\\b	Coincide con números enteros y decimales, e.g., 3, 5.67
OPERATOR	[+\\-*/=]	Coincide con operadores aritméticos y de asignación, e.g., +, =
PARENTHESIS	[()]	Coincide con paréntesis, e.g., (,)
VARIABLE	\\b[a-zA-Z]\\b	Coincide con nombres de variables, e.g., x, y, z
WHITESPACE	\\s+	Coincide con espacios en blanco (serán ignorados)

Reglas de coincidencia

02

Se uso la librería **re** quedando de la siguiente manera:

```
1 tokens = [  
2     ('COMENTARIO', r'\\/\\..*?\\/\\/'),           # Comentarios delimitados por //  
3     ('NUMERO_FLOTANTE', r'\b\d+\.\d+\b'),        # Números flotantes  
4     ('NUMERO_ENTERO', r'\b\d+\b'),               # Números enteros  
5     ('OPERADOR_ARITMETICO', r'[+\\-*/]'),        # Operadores aritméticos  
6     ('OPERADOR_RELACIONAL', r'==|!=|<=|<|>=|>'), # Operadores relacionales  
7     ('OPERADOR_LOGICO', r'&&|\\|\\|!'),          # Operadores lógicos  
8     ('ASIGNACION', r'='),                        # Operador de asignación  
9     ('PARENTESIS', r'[(\\)]'),                   # Paréntesis  
10    ('VARIABLE', r'\b[a-zA-Z]\b'),              # Variables (una sola letra)  
11    ('WHITESPACE', r'\\s+'),                     # Espacios en blanco  
12 ]
```

Descripción de tokens:

03

Números (NUMBER)

- Expresión Regular: `\b\d+(\.\d+)?\b`
- Explicación: Coincide con números enteros (`\d+`) y decimales (`\d+\.\d+`).

2. Operadores (OPERATOR)

- Expresión Regular: `[+\-*/=]`
- Explicación: Coincide con operadores aritméticos (+, -, *, /) y el operador de asignación (=)

3. Paréntesis (PARENTHESIS)

- Expresión Regular: `[()]`
- Explicación: Coincide con paréntesis izquierdo y derecho ((,)).

Descripción de tokens:

4. Variables (VARIABLE)

- Expresión Regular: `\b[a-zA-Z]\b`
- Explicación: Coincide con identificadores de una sola letra (x, y, z). Los identificadores son nombres para variables y deben ser una sola letra.

5. Espacios en Blanco (WHITESPACE)

- Expresión Regular: `\s+`
- Explicación: Coincide con espacios en blanco. Los espacios en blanco se ignoran durante el análisis léxico.

Documentación del analizador:

04

A continuación, se describe el funcionamiento y las partes del código del analizador léxico para álgebra básica. El analizador léxico es responsable de convertir una secuencia de caracteres en una secuencia de tokens, que son las unidades básicas de significado en el lenguaje.

$$\frac{x^a}{x^b} = x^{a-b}$$

Propósito: Esta sección define los tokens que el analizador léxico reconocerá en el código fuente.

- Detalles: Cada token está definido por un par de valores: un nombre y una expresión regular (regex). Las regex son utilizadas para identificar patrones específicos en el código fuente.

```
1  # Definición de tokens con prioridad
2  tokens = [
3      ('COMENTARIO', r'\\/\\..*?\\/\\.'),          # Comentarios delimitados por //
4      ('NUMERO_FLOTANTE', r'\b\d+\.\d+\b'),      # Números flotantes
5      ('NUMERO_ENTERO', r'\b\d+\b'),             # Números enteros
6      ('OPERADOR_ARITMETICO', r'[+|-|*|/|]'),    # Operadores aritméticos
7      ('OPERADOR_RELACIONAL', r'==|!=|<=|<|>=|>'), # Operadores relacionales
8      ('OPERADOR_LOGICO', r'&&|\\|\\|!'),        # Operadores lógicos
9      ('ASIGNACION', r'='),                     # Operador de asignación
10     ('PARENTESIS', r'[(|)]'),                 # Paréntesis
11     ('VARIABLE', r'\b[a-zA-Z]\b'),            # Variables (una sola letra)
12     ('WHITESPACE', r'\s+'),                   # Espacios en blanco
13 ]
```


Propósito: Esta función analiza el código fuente y genera una lista de tokens.

```
1  # Función para analizar el código fuente
2  def analizar_lexico(codigo):
3      tokens_encontrados = []
4      pos = 0
5      longitud_codigo = len(codigo)
6      while pos < longitud_codigo:
7          match = None
8
9          # Ignorar espacios en blanco
10         if re.match(tokens[-1][1], codigo[pos]):
11             pos += 1
12             continue
13
14         for tipo, patron in tokens:
15             regex = re.compile(patron)
16             match = regex.match(codigo, pos)
17             if match:
18                 valor = match.group(0)
19                 if tipo != 'WHITESPACE' and tipo != 'COMENTARIO': # Ignoramos espacios y comentarios
20                     tokens_encontrados.append((tipo, valor))
21                 pos = match.end(0)
22                 break
23             if not match:
24                 print(f"Error: Caracter no reconocido en la posición {pos}: '{codigo[pos]}'")
25                 return None
26
27         # Verificación de errores específicos
28         balance_parentesis = 0
29         for tipo, valor in tokens_encontrados:
30             if valor == '(':
31                 balance_parentesis += 1
32             elif valor == ')':
33                 balance_parentesis -= 1
34             if balance_parentesis < 0:
35                 print("Error: Paréntesis de cierre sin paréntesis de apertura.")
36                 return None
37
38         if balance_parentesis != 0:
39             print("Error: Paréntesis de apertura sin paréntesis de cierre.")
40             return None
41
42         return tokens_encontrados
```

Detalles:

- **Inicialización:** La lista `tokens_encontrados` almacena los tokens reconocidos. La variable `pos` mantiene la posición actual en el código fuente.
- **Bucle Principal:** Itera sobre el código fuente, carácter por carácter.

```
1  # Función para analizar el código fuente
2  def analizar_lexico(codigo):
3      tokens_encontrados = []
4      pos = 0
5      longitud_codigo = len(codigo)
6      while pos < longitud_codigo:
7          match = None
8
9          # Ignorar espacios en blanco
10         if re.match(tokens[-1][1], codigo[pos]):
11             pos += 1
12             continue
13
14         for tipo, patron in tokens:
15             regex = re.compile(patron)
16             match = regex.match(codigo, pos)
17             if match:
18                 valor = match.group(0)
19                 if tipo != 'WHITESPACE' and tipo != 'COMENTARIO': # Ignoramos espacios y comentarios
20                     tokens_encontrados.append((tipo, valor))
21                 pos = match.end(0)
22                 break
23             if not match:
24                 print(f"Error: Caracter no reconocido en la posición {pos}: '{codigo[pos]}'")
25                 return None
26
27         # Verificación de errores específicos
28         balance_parentesis = 0
29         for tipo, valor in tokens_encontrados:
30             if valor == '(':
31                 balance_parentesis += 1
32             elif valor == ')':
33                 balance_parentesis -= 1
34             if balance_parentesis < 0:
35                 print("Error: Paréntesis de cierre sin paréntesis de apertura.")
36                 return None
37
38         if balance_parentesis != 0:
39             print("Error: Paréntesis de apertura sin paréntesis de cierre.")
40             return None
41
42         return tokens_encontrados
```

- **Ignorar Espacios y Comentarios:** Espacios y comentarios se omiten.

- **Coincidencia de Tokens:** Se intenta hacer coincidir el código fuente con cada regex definida. Si se encuentra una coincidencia, se agrega a tokens_encontrados.

- **Errores de Paréntesis:** Se verifica el balance de paréntesis y se reportan errores si están desbalanceados.

```
1  # Función para analizar el código fuente
2  def analizar_lexico(codigo):
3      tokens_encontrados = []
4      pos = 0
5      longitud_codigo = len(codigo)
6      while pos < longitud_codigo:
7          match = None
8
9          # Ignorar espacios en blanco
10         if re.match(tokens[-1][1], codigo[pos]):
11             pos += 1
12             continue
13
14         for tipo, patron in tokens:
15             regex = re.compile(patron)
16             match = regex.match(codigo, pos)
17             if match:
18                 valor = match.group(0)
19                 if tipo != 'WHITESPACE' and tipo != 'COMENTARIO': # Ignoramos espacios y comentarios
20                     tokens_encontrados.append((tipo, valor))
21                 pos = match.end(0)
22                 break
23             if not match:
24                 print(f"Error: Caracter no reconocido en la posición {pos}: '{codigo[pos]}'")
25                 return None
26
27         # Verificación de errores específicos
28         balance_parentesis = 0
29         for tipo, valor in tokens_encontrados:
30             if valor == '(':
31                 balance_parentesis += 1
32             elif valor == ')':
33                 balance_parentesis -= 1
34             if balance_parentesis < 0:
35                 print("Error: Paréntesis de cierre sin paréntesis de apertura.")
36                 return None
37
38         if balance_parentesis != 0:
39             print("Error: Paréntesis de apertura sin paréntesis de cierre.")
40             return None
41
42         return tokens_encontrados
```

Propósito: Esta sección solicita al usuario que ingrese el código que desea analizar.

Detalles:


- **Entrada del Usuario:** El usuario ingresa el código línea por línea. El ingreso se termina al presionar Enter dos veces seguidas.

```
1  # Solicitar al usuario que ingrese el código
2  print("Ingresa tu código. Presiona Enter dos veces para finalizar.")
3  codigo_fuente = ""
4  while True:
5      linea = input()
6      if not linea:
7          break
8      codigo_fuente += linea + "\n"
```


Propósito: Esta sección llama a la función de análisis léxico y muestra los tokens encontrados.

Detalles:

- **Llamada a la Función:** Se pasa el código fuente ingresado por el usuario a la función `analizar_lexico`.
- **Mostrar Tokens:** Si se encuentran tokens, se imprimen en la salida.



```
1  # Llamar al analizador léxico
2  tokens_encontrados = analizar_lexico(codigo_fuente)
3  if tokens_encontrados:
4      for token in tokens_encontrados:
5          print(token)
6
```

Ejemplo de uso:

Ingresa tu código. Presiona Enter dos veces para finalizar.

```
(x+2)=4 ||hola||
```

```
('PARENTESIS', '(')
```

```
('VARIABLE', 'x')
```

```
('OPERADOR_ARITMETICO', '+')
```

```
('NUMERO_ENTERO', '2')
```

```
('PARENTESIS', ')')
```

```
('ASIGNACION', '=')
```

```
('NUMERO_ENTERO', '4')
```

```
PS D:\ProyectosPython>
```

Manejo de Errores

Descripción:

Propósito: El analizador léxico incluye manejo de errores específicos, como paréntesis desbalanceados y caracteres no reconocidos.

Detalles:

- Errores de Paréntesis: Se verifica que cada paréntesis de apertura tenga su correspondiente paréntesis de cierre.

```
PS D:\ProyectosPython> & C:/Users/adria/AppData/Local/Programs/Python/Python39-64/Scripts/python.exe  
Ingresa tu código. Presiona Enter dos veces para finalizar.  
(2+x=4  
  
Error: Paréntesis de apertura sin paréntesis de cierre.  
PS D:\ProyectosPython> |
```

Manejo de Errores

- **Caracteres no Reconocidos:** Si se encuentra un carácter que no coincide con ningún token definido, se informa un error con la posición del carácter en el código fuente.

```
PS D:\ProyectosPython> & C:/Users/adria/AppData/Local/Programs/Python/Python39-6/Scripts/python.exe C:/Users/adria/AppData/Local/Programs/Python/Python39-6/Scripts/python.exe
Ingresa tu código. Presiona Enter dos veces para finalizar
gd=3

Error: Caracter no reconocido en la posición 0: 'g'
PS D:\ProyectosPython> █
```