



TECNOLOGICO
NACIONAL DE MEXICO

SEP
SECRETARÍA DE
EDUCACIÓN PÚBLICA



ANALIZADOR LÉXICO

PF2024

Lenguajes Autómatas



Equipo:

Iker Daniel Moran Rodríguez

Angel Daniel Magaña Gutiérrez

Juan Pablo Jiménez Mendoza



11 DE ABRIL DE 2025
MARCO ANTONIO GUZMAN SOLANO
Instituto Tecnológico de Ciudad Guzmán

Índice

Introducción	2
Objetivos del Proyecto.....	2
Objetivo General	2
Objetivos Específicos	2
Diseño y Arquitectura	3
1. Limpieza de Comentarios.....	3
2. Explicación de instrucciones y E.R. (Expresiones Regulares)	3
3. Tokenización.....	7
Gramáticas	10
◆ Tipos de Tokens Reconocidos:.....	12
Manejo de Errores.....	12
3. Tabla de Símbolos	13
4. Archivos de Salida.....	13
Interfaz Gráfica (GUI).....	14
Prueba de Ejecución	15
Ventajas y Logros.....	22
Posibles Mejoras	22
Conclusiones Personales	22
Referencias bibliográficas.....	23



Reporte Técnico del Proyecto – Analizador Léxico PF2024

🔗 Introducción

En este proyecto desarrollé un **analizador léxico** personalizado utilizando el lenguaje de programación **Python**. El objetivo principal fue diseñar una herramienta capaz de **leer un archivo fuente de texto plano**, eliminar correctamente los comentarios sin alterar la estructura original del código, identificar y clasificar los componentes léxicos del lenguaje (tokens), y generar archivos de salida útiles para una futura fase de análisis sintáctico o semántico.

Además, implementé una **interfaz gráfica de usuario (GUI)** utilizando la librería Tkinter, con el fin de facilitar la interacción con la herramienta, permitiendo a cualquier usuario seleccionar archivos de entrada y recibir retroalimentación visual del análisis.

Este proyecto simula el funcionamiento de un compilador en su etapa léxica, tomando como base un lenguaje de programación tipo Pascal simplificado, utilizado comúnmente en ejercicios académicos.

⌚ Objetivos del Proyecto

Objetivo General

Desarrollar un analizador léxico funcional, modular y gráfico que permita procesar archivos fuente, reconocer tokens válidos y generar salidas auxiliares para una posible fase de compilación posterior.

Objetivos Específicos

- Eliminar comentarios sin alterar la estructura del código fuente.
- Identificar tokens léxicos válidos mediante expresiones regulares.
- Construir una tabla de símbolos única y ordenada.

- Generar archivos de salida .tok, .dep y .tab.
 - Implementar una interfaz gráfica amigable con el usuario.
-

Diseño y Arquitectura

1. Limpieza de Comentarios

Para comenzar el análisis, desarrollé la función limpiar_comentarios, que utiliza expresiones regulares para eliminar tanto comentarios de línea (//) como de bloque /* ... */. Un aspecto importante es que la eliminación se realiza **preservando los saltos de línea**, de forma que la numeración original del archivo fuente se mantenga intacta para el análisis posterior.

```
code = re.sub(r'//.*', "", code)  
code = re.sub(r'/*.*?\*/', "", code, flags=re.DOTALL)
```

Esto garantiza precisión al momento de reportar errores o mostrar tokens, ya que cada uno mantiene su línea original del código fuente.

2. Explicación de instrucciones y E.R. (Expresiones Regulares)

1. Palabras reservadas y tipos de datos

```
reserved = { ... } # Palabras clave como Inicio, Fin, Si...  
tipos_dato = { ... } # Tipos como Entero, Cadena, Booleano
```

Estas estructuras asocian una palabra del lenguaje a un **token personalizado**.

Ejemplo de uso en código fuente:

Inicio

 Se reconoce como PALABRA_RESERVADA_INICIO.

2. Lista de tokens

```
tokens = ['PROGRAMA', 'IDENTIFICADOR', 'CONSTANTE', ...]
```

Define **todos los posibles símbolos válidos y errores** que puede detectar el lexer.

3. E.R. simples (operadores y símbolos)

```
t_IGUAL = r'='
```

```
t_PAREN_IZQ = r'\('
```

```
t_PAREN_DER = r'\)'
```

```
t_SUMA = r'\+'
```

```
t_RESTA = r'-'
```

```
t_MULTIPLICACION = r'\*'
```

```
t_DIVISION = r'/'
```

```
t_ignore = '\t'
```

💡 Estas expresiones indican **símbolos individuales** como =, +, (, etc. El prefijo t_ es necesario para que PLY los registre como patrones.

💻 Ejemplo en código:

```
x = 10
```

➡ = es detectado como IGUAL.

4. Token de palabra clave: Script

```
def t_PROGRAMA(t):
```

```
    r'Script'
```

```
    t.type = 'PROGRAMA'
```

```
    return t
```

💡 Si se encuentra la palabra Script, se clasifica como inicio del programa.

5. Errores en identificadores

```
def t_ERROR_IDENTIFICADOR(t):
    r'[a-zA-Z_]*[^a-zA-Z0-9_\s()=-*/,:;"\n\t\r][a-zA-Z0-9_]*'
```

Este patrón detecta identificadores que contienen **caracteres no válidos** (como @ o #).

💻 Ejemplo:

mi@variable

➡ Será un ERROR_IDENTIFICADOR.

```
def t_ERROR_IDENTIFICADOR_NUM(t):
    r'\d+[a-zA-Z_]+[a-zA-Z0-9_]*'
```

Detecta identificadores que **empiezan con un número**, lo cual es inválido.

💻 Ejemplo:

123abc

➡ También se clasifica como error.

6. Identificadores válidos

```
def t_IDENTIFICADOR(t):
```

```
    r'[a-zA-Z_][a-zA-Z0-9_]*(\b|$)'
```

Detecta variables o nombres de funciones que empiezan con letra o guion bajo. Además, valida si no son palabras reservadas o tipos de datos.

💻 Ejemplo:

contador

➡ Se reconoce como IDENTIFICADOR.

7. Constantes numéricas

```
def t_CONSTANTE(t):
```

r'\d+'

Detecta números enteros.

▀ Ejemplo:

x = 123

→ 123 es una CONSTANTE.

8. Texto entre comillas (cadenas)

```
def t_TEXTO(t):
```

r'\".*?\\"

Captura texto rodeado de comillas dobles (").

▀ Ejemplo:

"Hola Mundo"

→ Se reconoce como TEXTO.

9. Comentarios

```
def t_COMENTARIO(t):
```

r'--.*'

Los comentarios empiezan con -- y se ignoran durante el análisis.

▀ Ejemplo:

-- Esto es un comentario

10. Manejo de líneas nuevas

```
def t_newline(t):
```

r'\n+'

```
t.lexer.lineno += len(t.value)
```

Lleva el conteo de líneas para poder registrar en qué renglón aparece cada token o error.

11. Manejo de errores generales

```
def t_error(t):
```

```
...
```

Este bloque detecta símbolos completamente **inválidos** que no coinciden con ninguna expresión regular definida.

■ Ejemplo:

```
@#$
```

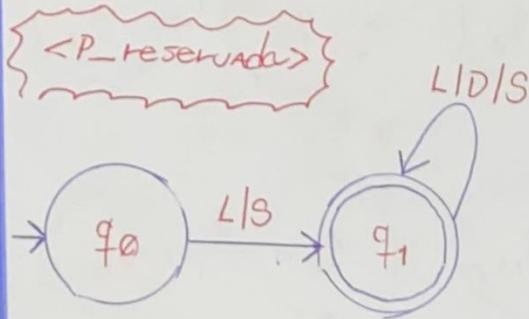
► Genera ERROR_LEXICO.

3. Tokenización

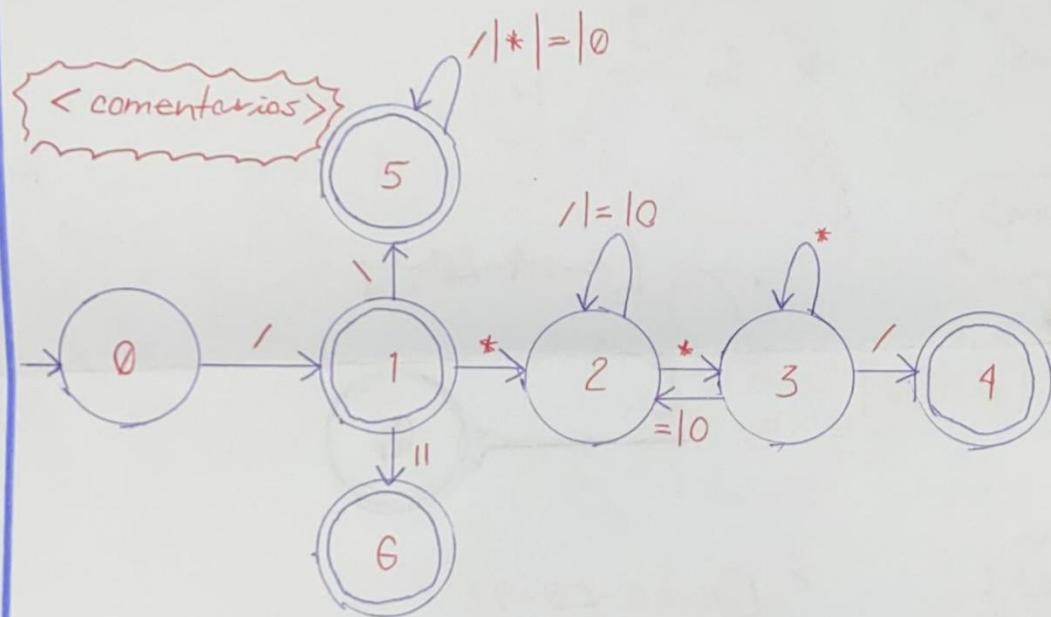
El núcleo del analizador es la función tokenize. Aquí definí un conjunto de patrones léxicos utilizando expresiones regulares para representar los diferentes elementos del lenguaje:

```
<US> ::= <p_reservada> | <s_simple> | <s_doble> | <id> | <cte_num>
<p_reservada> ::= begin | end | bool | int | ref | function | if | then | fi | else | while
| do | repeat | input | output | deref | true | false
<s_simple> ::= ; | , | + | - | * | ( | ) | = | >
<s_doble> ::= := | <=
<cte_num> ::= <dígito> | <cte_num> <dígito>
<id> ::= <letra> | <letra><resto_id>
<resto_id> ::= <alfanumérico> | <alfanumérico><resto_id>
<alfanumérico> ::= <dígito> | <letra>
<dígito> ::= 0 | 1 | ... | 9
<letra> ::= a | b | ... | z | A | B | ... | Z
```

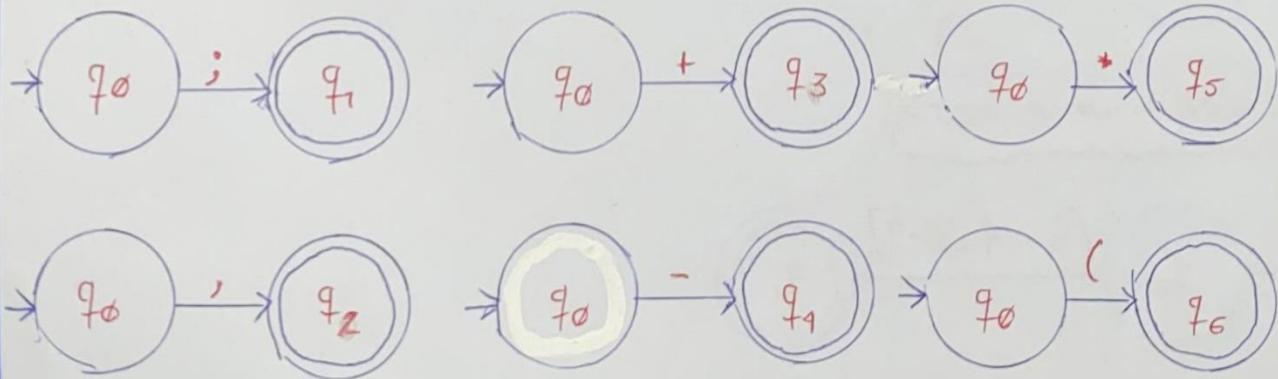
AUTOMATAS ANÁLISIS LÉXICO

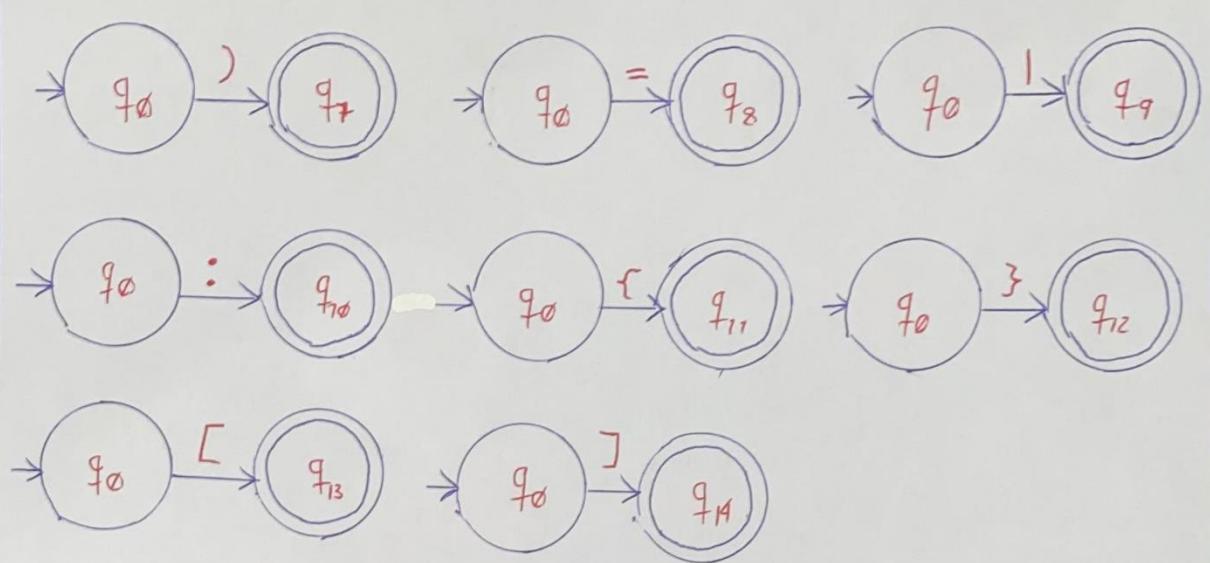


Q

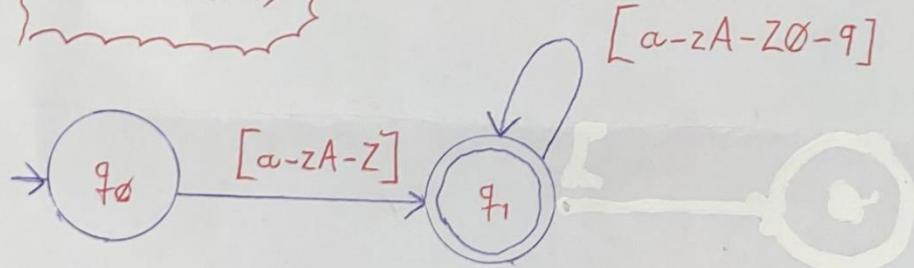


<g_simple>



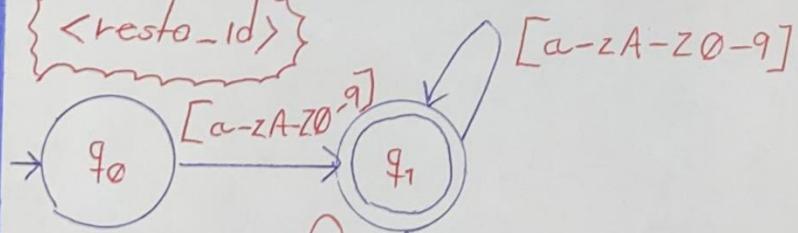


<cte-num>



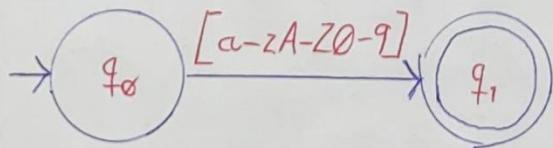
$[a-zA-Z0-9]$

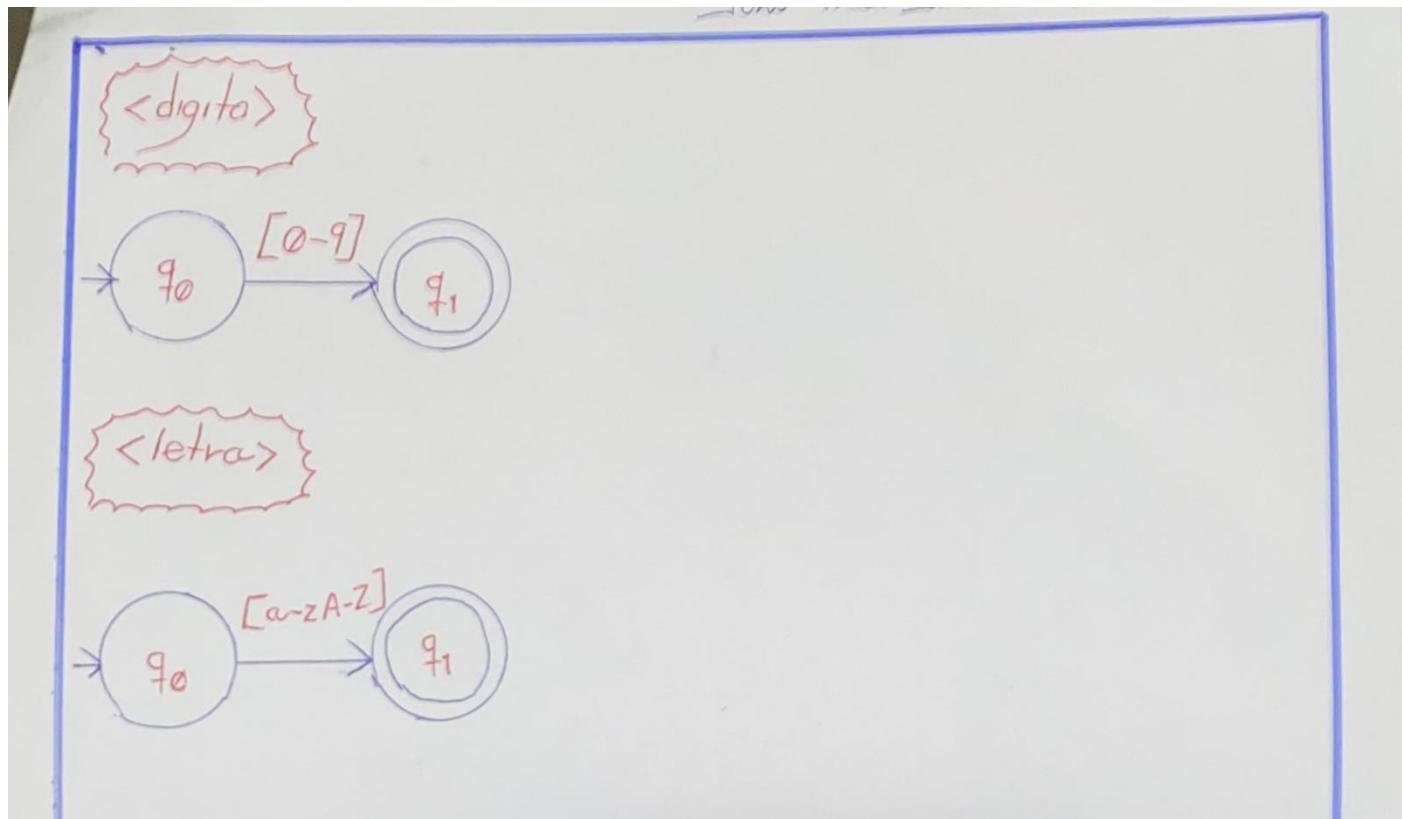
<resto-1d>



$[a-zA-Z0-9]$

<alfanumericos>





Gramáticas

En el desarrollo del analizador léxico, la gramática no se define de forma completa en esta etapa, ya que la responsabilidad del análisis léxico es únicamente **identificar los componentes léxicos (tokens)** del lenguaje, como palabras clave, identificadores, operadores, literales, etc.

Sin embargo, para una comprensión integral del funcionamiento del analizador, es útil establecer una **gramática léxica** que describa las expresiones regulares utilizadas para reconocer cada tipo de token.

abc Gramática léxica del lenguaje (fragmento)

A continuación, se muestran algunas reglas léxicas utilizadas por el analizador:

- **Identificadores:**

$ID \rightarrow [a-zA-Z_][a-zA-Z0-9_]^*$

Secuencias que comienzan con una letra o guion bajo, seguidas de letras, dígitos o guiones bajos.

- **Números enteros:**

INT → [0-9]+

Uno o más dígitos numéricos.

- **Palabras clave:**

Reconocidas mediante coincidencias directas:

IF → 'if'

ELSE → 'else'

WHILE → 'while'

- **Operadores:**

PLUS → '+'

MINUS → '-'

TIMES → '*'

DIVIDE → '/'

- **Símbolos especiales y delimitadores:**

LPAREN → '('

RPAREN → ')'

LBRACE → '{'

RBRACE → '}'

SEMICOLON → ';

Estas reglas permiten que el analizador léxico construido con **PLY (Python Lex-Yacc)** identifique correctamente los tokens, generando un archivo .tok con el flujo de símbolos que serán utilizados posteriormente en el análisis sintáctico.

◊ Tipos de Tokens Reconocidos:

- **Palabras Reservadas:** decl, int, Cad, Bool, Inicio, Fin, impcad, leerdig
- **Operadores y símbolos:** :=, +, -, *, /, ,, ;, (), {}, etc.
- **Constantes:** Números enteros, cadenas entre comillas dobles " o curvas ""
- **Identificadores:** Palabras con letras, números y guiones bajos
- **Espacios y saltos de línea:** manejados para mantener legibilidad sin generar tokens

Cada token es almacenado junto con su número de línea, lexema y código, que posteriormente se convierte en una descripción textual amigable para los archivos de salida.

⚠ Manejo de Errores

Si se encuentra un carácter no válido, se clasifica como ERROR con código 999, permitiendo detectar anomalías léxicas rápidamente.

```
def t_ERROR_IDENTIFICADOR(t):
    r'[a-zA-Z_]*[a-zA-Z0-9_\s()=-*/;:"\n\t\r][a-zA-Z0-9_]*'
    # Nota el añadido de la coma en los caracteres excluidos ^
    errores_lexicos.append({
        'line': t.lineno,
        'value': t.value,
        'type': 'ERROR_IDENTIFICADOR'
    })
    return t

def t_ERROR_IDENTIFICADOR_NUM(t):
    r'\d+[a-zA-Z_]+[a-zA-Z0-9_]*'
    errores_lexicos.append({
        'line': t.lineno,
        'value': t.value,
        'type': 'ERROR_IDENTIFICADOR'
    })
    return t

for i, simbolo in enumerate(simbolos, 1):
    f.write("{:<8} {:<20} {:<50} {:<15}\n".format(
        i,
        simbolo['value'],
        simbolo['type'],
        token_codes.get(simbolo['type'], 999)))
```

3. Tabla de Símbolos

Durante la tokenización, simultáneamente construyo una tabla de símbolos que almacena:

```
with open('progfte.tab', 'w', encoding='utf-8') as f:  
    f.write("{:<8} {:<20} {:<50} {:<15}\n".format(  
        "No", "Lexema", "Token", "Referencia"))  
    f.write("-"*93 + "\n")
```

- **Número de entrada**
- **Lexema**
- **Tipo de símbolo** (Identificador, constante, palabra reservada)
- **Referencia única**

Este registro es muy útil para etapas posteriores como análisis semántico o generación de código intermedio. Se evita el registro duplicado de símbolos mediante un set.

4. Archivos de Salida

Una vez procesado el archivo fuente, se generan tres salidas:

 **.tok – Archivo de Tokens**

Contiene una lista formateada con cada token detectado:

Renglón: 4, Lexema: impcad , Token: 616, Palabra reservada (impcad)

 **.dep – Código Depurado**

Contiene una versión compacta del código, sin comentarios ni espacios innecesarios. Esto permite observar únicamente la lógica "real" del programa.

 **.tab – Tabla de Símbolos**

Formateada en columnas, presenta cada símbolo único con su referencia correspondiente:

No LEXEMA	TOKEN	REF
-----	-----	-----
1 posición	Identificador	100
2 a	Identificador	101

⌚ Interfaz Gráfica (GUI)

La GUI fue diseñada con Tkinter, utilizando una estructura limpia y profesional:

- **Título personalizado:** “Analizador Léxico PF2024”
- **Botón para selección de archivo:** Permite cargar un .txt desde el explorador del sistema
- **Indicador de estado:** Muestra mensajes de éxito o error después del análisis
- **Panel de imagen:** Prepara espacio para incluir un logo o imagen representativa del lenguaje

Este diseño hace que la herramienta sea **intuitiva incluso para usuarios no técnicos**, brindando una experiencia más accesible y visual.



Prueba de Ejecución

Utilicé el siguiente código de ejemplo:

```
pf2024 Ejem1          --Encabezado, deberá llevar el
nombre del lenguaje--



int número = 1
int posicion

de@cl           --Pal_Reserv Declaración de variables--
--Comentario--
int posición a,b,c,f  ;
Cad d;
Bool e;

Inicio          --Cuerpo del programa,
inicio etc--
a:=5;
b:=3;
impcad("Dame un numero, Hola mundo");
--solicita un número--
leerdig(var);      --Guarda un valor entero en una variable--
d:=a*b-c;
d:=a* (b-c);
impcad(d);
Fin
```

✓ Resultados Obtenidos:

Archivo de Tokens

Renglón: 1 Lexema: pf2024 Token: IDENTIFICADOR

Renglón: 1 Lexema: Ejem1 Token: IDENTIFICADOR

Renglón: 3 Lexema: int Token: IDENTIFICADOR

Renglón: 3	Lexema: número	Token: ERROR_IDENTIFICADOR
Renglón: 3	Lexema: =	Token: IGUAL
Renglón: 3	Lexema: 1	Token: CONSTANTE
Renglón: 4	Lexema: int	Token: IDENTIFICADOR
Renglón: 4	Lexema: pos+cion	Token: ERROR_IDENTIFICADOR
Renglón: 6	Lexema: de@cl	Token: ERROR_IDENTIFICADOR
Renglón: 8	Lexema: int	Token: IDENTIFICADOR
Renglón: 8	Lexema: posicion	Token: IDENTIFICADOR
Renglón: 8	Lexema: a	Token: IDENTIFICADOR
Renglón: 8	Lexema: ,	Token: COMA
Renglón: 8	Lexema: b	Token: IDENTIFICADOR
Renglón: 8	Lexema: ,	Token: COMA
Renglón: 8	Lexema: c	Token: IDENTIFICADOR
Renglón: 8	Lexema: ,	Token: COMA
Renglón: 8	Lexema: f	Token: IDENTIFICADOR
Renglón: 8	Lexema: ;	Token: PuntoYComa
Renglón: 9	Lexema: Cad	Token: IDENTIFICADOR
Renglón: 9	Lexema: d	Token: IDENTIFICADOR
Renglón: 9	Lexema: ;	Token: PuntoYComa
Renglón: 10	Lexema: Bool	Token: IDENTIFICADOR
Renglón: 10	Lexema: e	Token: IDENTIFICADOR
Renglón: 10	Lexema: ;	Token: PuntoYComa
Renglón: 12	Lexema: Inicio	Token: PALABRA_RESERVADA_INICIO
Renglón: 13	Lexema: a	Token: IDENTIFICADOR
Renglón: 13	Lexema: :	Token: DosPuntos
Renglón: 13	Lexema: =	Token: IGUAL

Renglón: 13	Lexema: 5	Token: CONSTANTE
Renglón: 13	Lexema: ;	Token: PuntoYComa
Renglón: 14	Lexema: b	Token: IDENTIFICADOR
Renglón: 14	Lexema: :	Token: DosPuntos
Renglón: 14	Lexema: =	Token: IGUAL
Renglón: 14	Lexema: 3	Token: CONSTANTE
Renglón: 14	Lexema: ;	Token: PuntoYComa
Renglón: 15	Lexema: impcad	Token: IDENTIFICADOR
Renglón: 15	Lexema: (Token: PAREN_IZQ
Renglón: 15	Lexema: Dame un numero, Hola mundo	Token: TEXTO
Renglón: 15	Lexema:)	Token: PAREN_DER
Renglón: 15	Lexema: ;	Token: PuntoYComa
Renglón: 16	Lexema: leerdig	Token: IDENTIFICADOR
Renglón: 16	Lexema: (Token: PAREN_IZQ
Renglón: 16	Lexema: var	Token: IDENTIFICADOR
Renglón: 16	Lexema:)	Token: PAREN_DER
Renglón: 16	Lexema: ;	Token: PuntoYComa
Renglón: 17	Lexema: d	Token: IDENTIFICADOR
Renglón: 17	Lexema: :	Token: DosPuntos
Renglón: 17	Lexema: =	Token: IGUAL
Renglón: 17	Lexema: a	Token: IDENTIFICADOR
Renglón: 17	Lexema: *	Token: MULTIPLICACION
Renglón: 17	Lexema: b	Token: IDENTIFICADOR
Renglón: 17	Lexema: -	Token: RESTA
Renglón: 17	Lexema: c	Token: IDENTIFICADOR
Renglón: 17	Lexema: ;	Token: PuntoYComa

Renglón: 18	Lexema: d	Token: IDENTIFICADOR
Renglón: 18	Lexema: :	Token: DosPuntos
Renglón: 18	Lexema: =	Token: IGUAL
Renglón: 18	Lexema: a	Token: IDENTIFICADOR
Renglón: 18	Lexema: *	Token: MULTIPLICACION
Renglón: 18	Lexema: (Token: PAREN_IZQ
Renglón: 18	Lexema: b	Token: IDENTIFICADOR
Renglón: 18	Lexema: -	Token: RESTA
Renglón: 18	Lexema: c	Token: IDENTIFICADOR
Renglón: 18	Lexema:)	Token: PAREN_DER
Renglón: 18	Lexema: ;	Token: PuntoYComa
Renglón: 19	Lexema: impcad	Token: IDENTIFICADOR
Renglón: 19	Lexema: (Token: PAREN_IZQ
Renglón: 19	Lexema: d	Token: IDENTIFICADOR
Renglón: 19	Lexema:)	Token: PAREN_DER
Renglón: 19	Lexema: ;	Token: PuntoYComa
Renglón: 20	Lexema: Fin	Token: PALABRA_RESERVADA_FIN

Tabla de Símbolos

No	Lexema	Token	Referencia
<hr/>			
1	pf2024	IDENTIFICADOR	300
2	Ejem1	IDENTIFICADOR	300
3	int	IDENTIFICADOR	300
5	=	IGUAL	70
6	1	CONSTANTE	400

7	int	IDENTIFICADOR	300
10	int	IDENTIFICADOR	300
11	posicion	IDENTIFICADOR	300
12	a	IDENTIFICADOR	300
13	,	COMA	80
14	b	IDENTIFICADOR	300
15	,	COMA	80
16	c	IDENTIFICADOR	300
17	,	COMA	80
18	f	IDENTIFICADOR	300
19	;	PuntoYComa	81
20	Cad	IDENTIFICADOR	300
21	d	IDENTIFICADOR	300
22	;	PuntoYComa	81
23	Bool	IDENTIFICADOR	300
24	e	IDENTIFICADOR	300
25	;	PuntoYComa	81
26	Inicio	PALABRA_RESERVADA_INICIO	1
27	a	IDENTIFICADOR	300
28	:	DosPuntos	82
29	=	IGUAL	70
30	5	CONSTANTE	400
31	;	PuntoYComa	81
32	b	IDENTIFICADOR	300
33	:	DosPuntos	82
34	=	IGUAL	70

35	3	CONSTANTE	400
36	;	PuntoYComa	81
37	impcad	IDENTIFICADOR	300
38	(PAREN_IZQ	50
39	Dame un numero, Hola mundo TEXTO		500
40)	PAREN_DER	51
41	;	PuntoYComa	81
42	leerdig	IDENTIFICADOR	300
43	(PAREN_IZQ	50
44	var	IDENTIFICADOR	300
45)	PAREN_DER	51
46	;	PuntoYComa	81
47	d	IDENTIFICADOR	300
48	:	DosPuntos	82
49	=	IGUAL	70
50	a	IDENTIFICADOR	300
51	*	MULTIPLICACION	62
52	b	IDENTIFICADOR	300
53	-	RESTA	61
54	c	IDENTIFICADOR	300
55	;	PuntoYComa	81
56	d	IDENTIFICADOR	300
57	:	DosPuntos	82
58	=	IGUAL	70
59	a	IDENTIFICADOR	300
60	*	MULTIPLICACION	62

61	(PAREN_IZQ	50
62	b	IDENTIFICADOR	300
63	-	RESTA	61
64	c	IDENTIFICADOR	300
65)	PAREN_DER	51
66	;	PuntoYComa	81
67	impcad	IDENTIFICADOR	300
68	(PAREN_IZQ	50
69	d	IDENTIFICADOR	300
70)	PAREN_DER	51
71	;	PuntoYComa	81
72	Fin	PALABRA_RESERVADA_FIN	2

Código Depurado

pf2024Ejem1intnúmero=1intpos+cionde@clintposiciona,
 f;Cadd;Boole;Inicioa:=5;b:=3;impcad(Dame un numero,
 mundo);leerdig(var);d:=a*b-c;d:=a*(b-c);impcad(d);Fin
 Hola

- Todos los comentarios fueron eliminados correctamente sin alterar los renglones.
 - Los tokens fueron reconocidos con precisión (identificadores, operadores, cadenas, constantes numéricas).
 - Los archivos .tok, .dep, y .tab se generaron exitosamente.
 - No se detectaron errores léxicos en la entrada.
-

Ventajas y Logros

- ✓ Precisión en el conteo de líneas gracias al control de saltos.
 - ✓ Separación modular del código (limpieza, tokenización, archivos, interfaz).
 - ✓ Generación profesional de tablas de símbolos.
 - ✓ Interfaz amigable para facilitar el uso sin depender del entorno de desarrollo.
-

Posibles Mejoras

- Agregar soporte para más palabras reservadas y estructuras del lenguaje.
 - Implementar visualización integrada de los archivos .tok, .tab y .dep.
 - Extender la funcionalidad hacia un **analizador sintáctico**.
 - Permitir edición en vivo del código fuente desde la misma interfaz.
 - Añadir opción para exportar todo el análisis en PDF o Excel.
-

Conclusiones Personales

Juan Pablo Jimenez Mendoza

Desarrollar este proyecto me permitió comprender de forma práctica cómo un lenguaje de programación es interpretado desde sus unidades más básicas. Ver cómo las expresiones regulares se transforman en tokens me ayudó a visualizar el papel fundamental que cumple el análisis léxico en los compiladores.

Iker Daniel Moran Rodríguez

Durante la carrera hemos estudiado conceptos como gramáticas, autómatas y expresiones regulares, pero esta fue una oportunidad para aplicarlos en un entorno real, utilizando herramientas como PLY y Python, lo que reforzó mis conocimientos de forma significativa.

Ángel Daniel Magaña Gutiérrez

La estructura del programa, la separación por archivos .tok, .tab y .dep, y la

integración con una interfaz gráfica evidencian la necesidad de mantener una arquitectura clara y modular al desarrollar herramientas de análisis de código.

Referencias bibliográficas

1. Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). *Compiladores: Principios, técnicas y herramientas* (2.^a ed.). Pearson Educación.
2. Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2007). *Introducción a la teoría de autómatas, lenguajes y computación* (3.^a ed.). Pearson Educación.
3. Python Software Foundation. (2024). *Python 3 Documentation*.
4. Tkinter. (2024). *Tkinter – Python Interface to Tcl/Tk*.