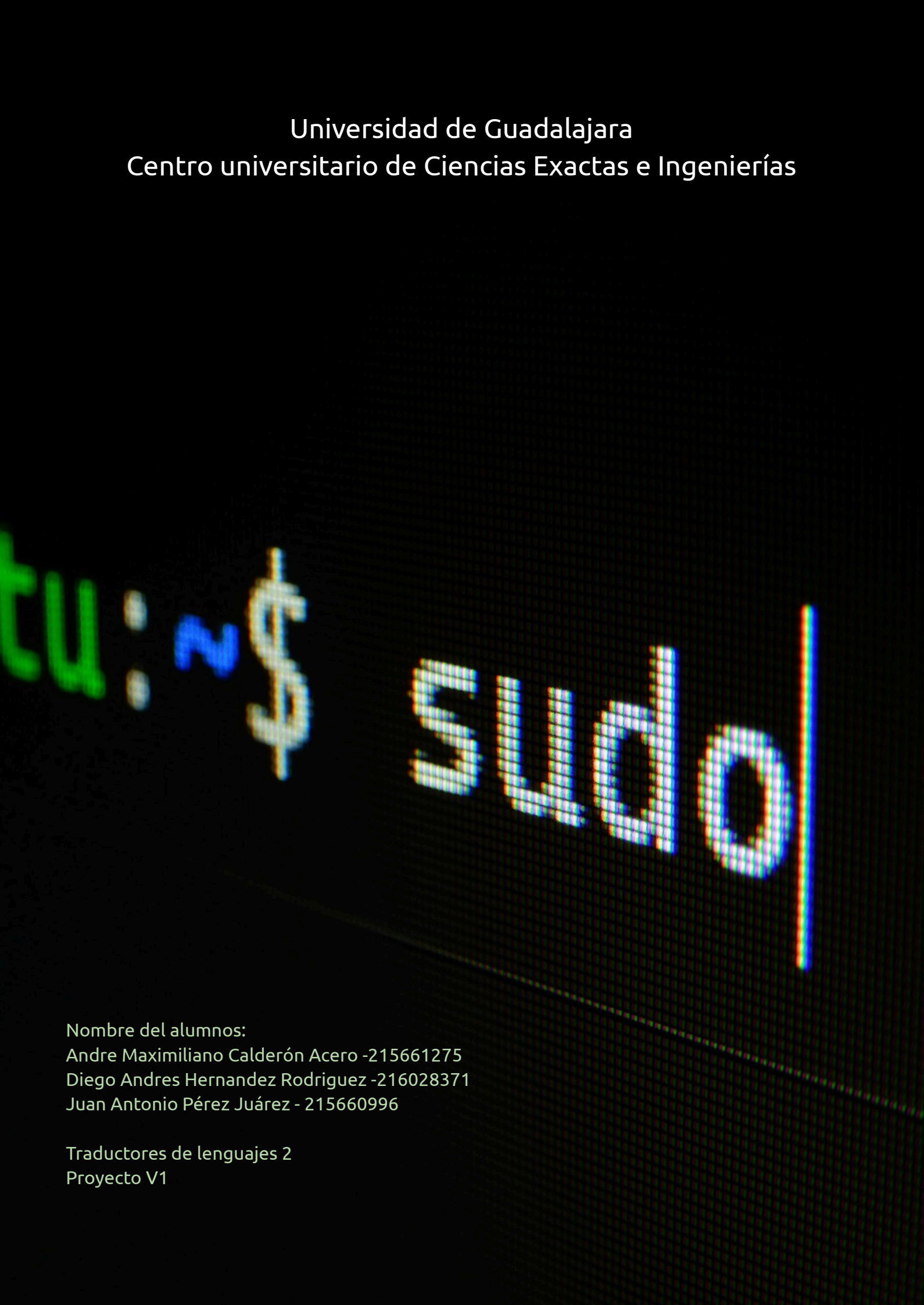
Universidad de Guadalajara
Centro universitario de Ciencias Exactas e Ingenierías



Nombre del alumnos:
Andre Maximiliano Calderón Acero -215661275
Diego Andres Hernandez Rodriguez -216028371
Juan Antonio Pérez Juárez - 215660996

Traductores de lenguajes 2
Proyecto V1

Actividad

Implementar un analizador sintáctico descendente de la gramática del lenguaje de programación corregido en la práctica 3.

El programa recibe un archivo fuente de entrada. De salida solo nos va indicar si el programa fuente es válido de acuerdo a la gramática ó, en caso contrario mostrar el mensaje de error correspondiente.

Marco Teórico

Es importante recalcar que para esta práctica nos sirve el hecho de tener la teoria de la actividad pasada, por que sigue siendo relevante para nuestro proyecto, asi que no está de más el repasarlo nuevamente.

Un analizador sintáctico descendente predictivo es un tipo de parser que procesa la entrada desde el símbolo inicial de la gramática hacia las hojas, utilizando un enfoque predictivo basado en gramáticas LL(1), lo que evita el retroceso mediante el uso de una tabla de análisis que determina de manera única la producción a aplicar en cada paso. Este analizador trabaja en conjunto con un analizador léxico, el cual se encarga de leer el flujo de caracteres de entrada y agruparlos en tokens como identificadores (definidos por patrones como [a-zA-Z_][a-zA-Z0-9_]*) y números de más de un dígito (enteros o decimales, reconocidos mediante expresiones como [0-9]+(\.[0-9]+)?).

La gramática debe estar diseñada para ser no ambigua y sin recursión por la izquierda, permitiendo así una derivación eficiente. El proceso de análisis sintáctico se basa en comparar los tokens generados por el lexer con las reglas gramaticales almacenadas en la tabla predictiva, expandiendo los no terminales según corresponda y construyendo el árbol de derivación de manera sistemática. Este enfoque es eficiente en tiempo lineal, aunque presenta limitaciones al requerir gramáticas LL(1), lo que puede exigir modificaciones en la definición original del lenguaje. Su aplicación es común en el desarrollo de intérpretes y compiladores para lenguajes con estructuras sintácticas bien definidas y relativamente simples.

1) E → E1 + T	E.nodo = new Nodo('+', E1.nodo, T.nodo)
2) E → E1 - T	E.nodo = new Nodo('-', E1.nodo, T.nodo)
3) E → T	E.nodo = T.nodo
4) T → T1 * F	T.nodo = new Nodo('*', T1.nodo, F.nodo)
5) T → T1 / F	T.nodo = new Nodo('/', T1.nodo, F.nodo)
6) T → F	T.nodo = F.nodo
7) F → (E)	F.nodo = E.nodo
8) F → id	F.nodo = new Hoja(id, id.entrada)
9) F → num	F.nodo = new Hoja(num, num.val)

Código en python de la práctica

Python

```python
import tkinter as tk
from tkinter import filedialog
from tkinter import ttk
from lexer import Lexer

class LexicalAnalyzerGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("Analizador")
        self.root.configure(bg='#f0f0f0')  # Color de fondo suave

        # Configuración de estilos
        style = ttk.Style()
        style.configure("Treeview",
                        background="#ffffff",
                        fieldbackground="#ffffff",
                        foreground="#333333")
        style.configure("Treeview.Heading",
                        background="#e1e1e1",
                        font=('Arial', 9, 'bold'))
        style.configure("TButton",
                        padding=5,
                        background="#4a90e2")

        # Frame principal con padding
        main_frame = ttk.Frame(self.root, padding="10")
        main_frame.pack(fill=tk.BOTH, expand=True)

        # Área de texto con borde y fondo
        self.text_area = tk.Text(
            main_frame,
            wrap="word",
            width=50,
            height=10,
            font=('Consolas', 10),
            bg='white',
            relief="solid",
            borderwidth=1
```

```
        )
        self.text_area.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)
        self.text_area.insert(tk.END, "a = 7\nb = 2\nsuma = a+b\nresta =
ab\nprint('suma:',suma, '\nResta:',resta )\n\n")

        # Frame para botones
        button_frame = ttk.Frame(main_frame)
        button_frame.pack(pady=5)

        # Botones estilizados
        self.read_button = ttk.Button(
            button_frame,
            text="Leer archivo",
            command=self.read_file,
            style="TButton"
        )
        self.read_button.pack(side=tk.LEFT, padx=5)

        self.analyze_button = ttk.Button(
            button_frame,
            text="Analizar léxicamente",
            command=self.analyze_lexically,
            style="TButton"
        )
        self.analyze_button.pack(side=tk.LEFT, padx=5)

        self.create_table()

        # Frame para la tabla semántica
        self.semantic_frame = ttk.Frame(main_frame)
        self.semantic_frame.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)

        # Tabla semántica
        self.semantic_tree = ttk.Treeview(
            self.semantic_frame,
            columns=("Lexma", "Token", "Coincidencia"),
            show="headings",
            height=10
        )

        # Configuración de columnas
        for col in ("Lexma", "Token", "Coincidencia"):
            self.semantic_tree.heading(col, text=col)
```

```
            self.semantic_tree.column(col, width=150)

        self.semantic_tree.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

        # Scrollbar para tabla semántica
        self.semantic_scrollbar = ttk.Scrollbar(
            self.semantic_frame,
            orient="vertical",
            command=self.semantic_tree.yview
        )
        self.semantic_scrollbar.pack(side=tk.RIGHT, fill="y")

self.semantic_tree.configure(yscrollcommand=self.semantic_scrollbar.set)

    def create_table(self):
        # Frame para la primera tabla
        self.table_frame = ttk.Frame(self.root)
        self.table_frame.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)

        # Primera tabla
        self.tree = ttk.Treeview(
            self.table_frame,
            columns=("Lexma", "Token"),
            show="headings",
            height=10
        )

        # Configuración de columnas
        for col in ("Lexma", "Token"):
            self.tree.heading(col, text=col)
            self.tree.column(col, width=150)

        self.tree.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

        # Scrollbar para primera tabla
        self.scrollbar = ttk.Scrollbar(
            self.table_frame,
            orient="vertical",
            command=self.tree.yview
        )
        self.scrollbar.pack(side=tk.RIGHT, fill="y")
        self.tree.configure(yscrollcommand=self.scrollbar.set)
```

```
    def update_table(self, tokens):
        for token in tokens:
            lexma, token_name = token[1], token[0]
            self.tree.insert("", "end", values=(lexma, token_name))

    def read_file(self):
        file_path = filedialog.askopenfilename(filetypes=[("Text files",
"*.txt")])
        if file_path:
            with open(file_path, 'r') as file:
                content = file.read()
                self.text_area.delete("1.0", tk.END)
                self.text_area.insert(tk.END, content)

    def analyze_lexically(self):
        input_string = self.text_area.get("1.0", tk.END)
        lexer = Lexer(input_string)
        tokens = lexer.lex()

        self.tree.delete(*self.tree.get_children())

        for token in tokens:
            lexma, token_name = token[1], token[0]
            self.tree.insert("", "end", values=(lexma, token_name))

        self.compare_with_file(tokens)

    def compare_with_file(self, tokens_input):
        file_path = filedialog.askopenfilename(filetypes=[("Text files",
"*.txt")])
        if file_path:
            with open(file_path, 'r') as file:
                content = file.read()
                lexer = Lexer(content)
                tokens_file = lexer.lex()

                self.semantic_tree.delete(*self.semantic_tree.get_children())

                for token_input, token_file in zip(tokens_input, tokens_file):
                    lexma_input, token_name_input = token_input[1],
token_input[0]
                    lexma_file, token_name_file = token_file[1], token_file[0]
```

```
                    coincidence = "✓" if lexma_input == lexma_file and
token_name_input == token_name_file else "✗"

                    # Insertar con tags para colorear
                    tag = "valid" if coincidence == "✓" else "invalid"
                    self.semantic_tree.insert("", "end",
                                              values=(lexma_input,
token_name_input, coincidence),
                                              tags=(tag,))

                # Configurar colores para las filas
                self.semantic_tree.tag_configure("valid", background="#e6ffe6")
                self.semantic_tree.tag_configure("invalid",
background="#ffe6e6")

if __name__ == "__main__":
    root = tk.Tk()
    app = LexicalAnalyzerGUI(root)
    root.mainloop()
```

Lógica del programa

Se crea una clase que encapsula toda la funcionalidad del analizador.
El método __init__ configura la ventana principal y todos sus componentes.

```Python
class LexicalAnalyzerGUI:
    def __init__(self, root):
        # Configuración inicial
```

Se establece el título de la ventana.
Se configuran los colores de fondo y estilos visuales para mejorar la apariencia
Se establecen los estilos de los widgets ttk (tablas, botones, etc.)

```Python
self.root.title("Analizador")
self.root.configure(bg='#f0f0f0')
style = ttk.Style()
# Configuraciones de estilo...
```

Se crea un área de texto donde el usuario puede escribir o ver el código a analizar
Se configura propiedades como fuente, tamaño y color
Se inserta un texto de ejemplo predeterminado

```python
self.text_area = tk.Text(...)
self.text_area.pack(...)
self.text_area.insert(...)
```

Se crea un marco para contener los botones
Se añade dos botones principales: uno para leer archivos y otro para analizar
Conectar cada botón con su función correspondiente

```python
button_frame = ttk.Frame(main_frame)
self.read_button = ttk.Button(...)
self.analyze_button = ttk.Button(...)
```

Se implementa un método para crear la primera tabla (treeview)
Esta tabla mostrará los tokens encontrados en el análisis léxico
Se configura columnas para mostrar lexema y token
Se añade scrollbar para navegar por resultados extensos

```python
def create_table(self):
    # Configuración de la tabla...
```

Se crea una segunda tabla para mostrar comparaciones
Esta tabla incluye columnas adicionales para indicar coincidencias
Se configura esta tabla con su propio scrollbar

```python
self.semantic_tree = ttk.Treeview(...)
# Configuración de la tabla semántica...
```

Se implementa la funcionalidad para abrir un diálogo de selección de archivos
Se lee el contenido del archivo seleccionado
Actualiza el área de texto con el contenido leído

```python
def read_file(self):
    file_path = filedialog.askopenfilename(...)
    # Lectura del archivo...
```

Se obtiene el texto del área de texto
Crea una instancia del analizador léxico
Procesa el texto para obtener tokens

Actualiza la tabla con los resultados del análisis

```python
def analyze_lexically(self):
    input_string = self.text_area.get("1.0", tk.END)
    lexer = Lexer(input_string)
    tokens = lexer.lex()
    # Actualización de la tabla...
```

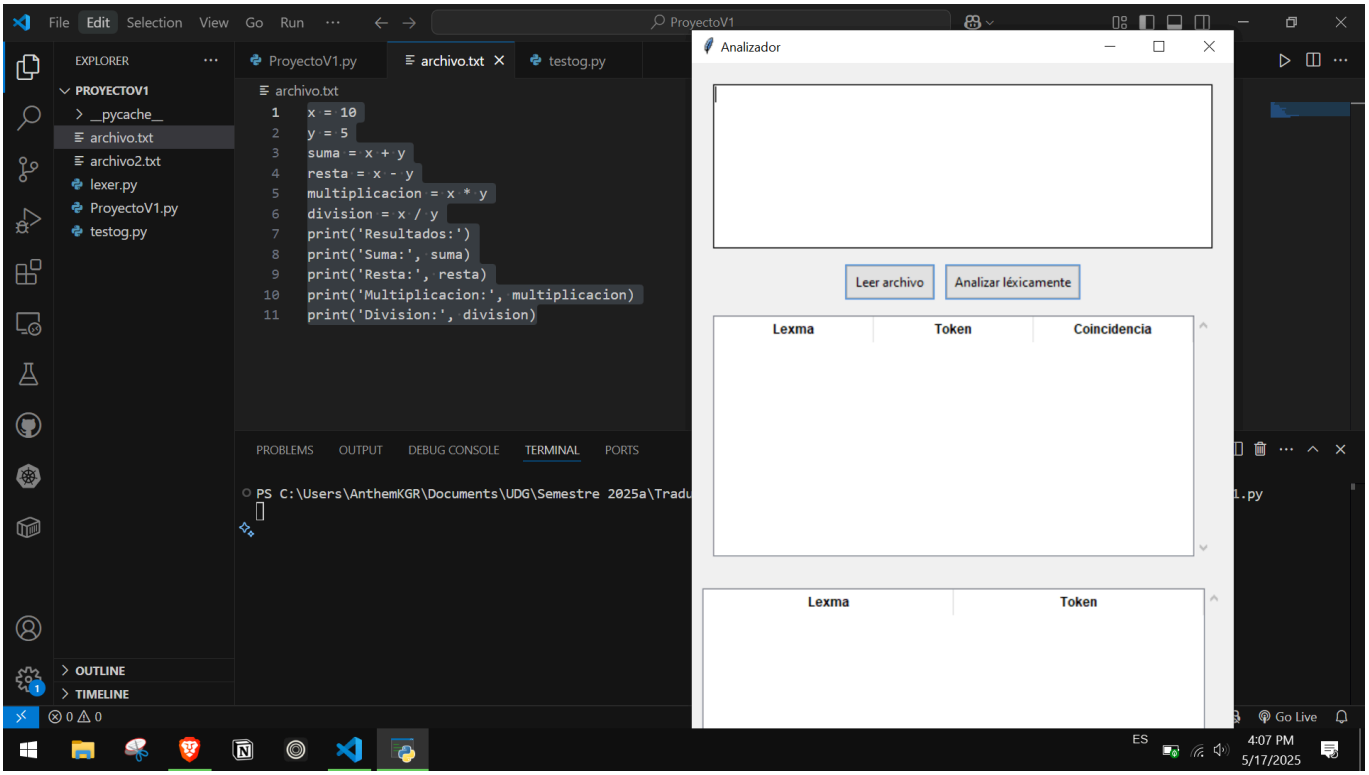
Permite seleccionar un segundo archivo para comparar
Realiza el análisis léxico de este archivo
Compara los tokens de ambos análisis
Muestra los resultados en la tabla semántica con indicadores visuales

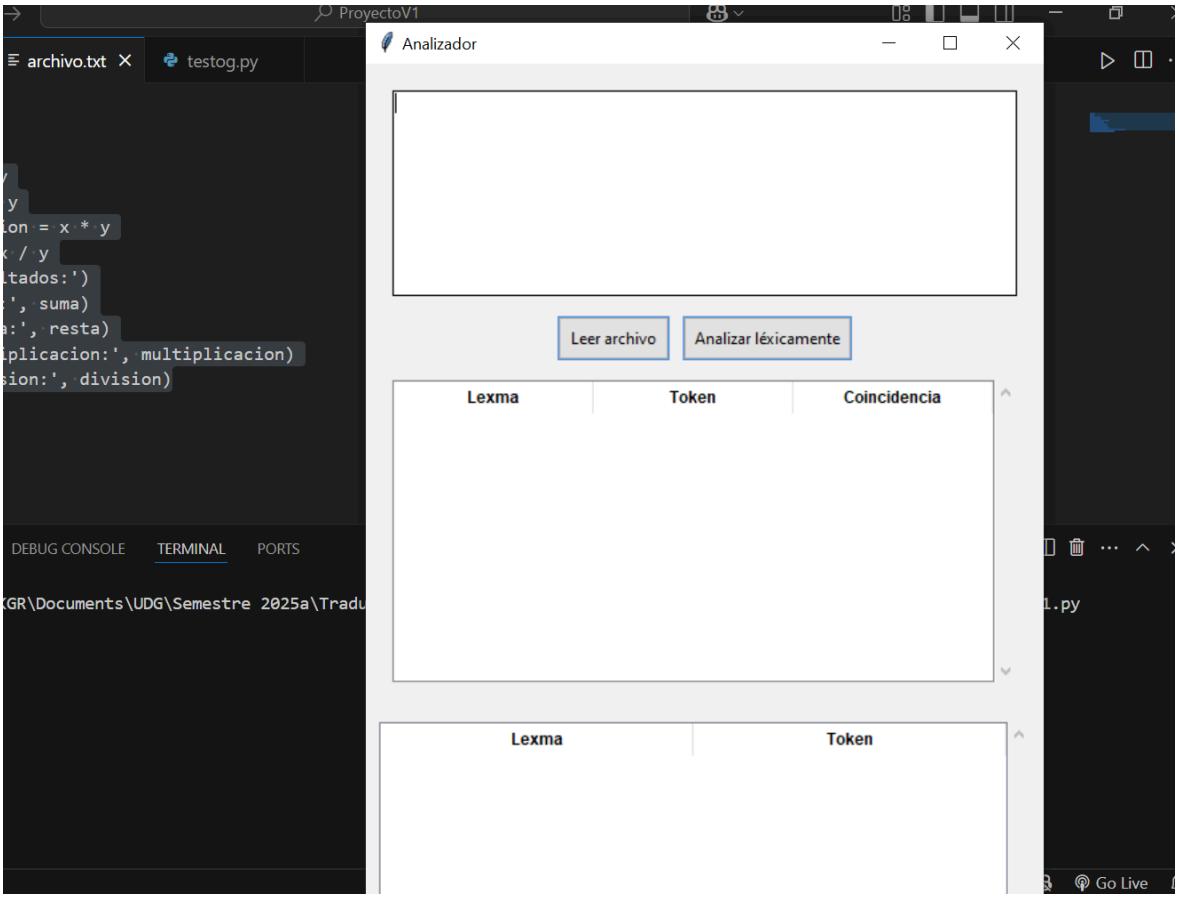
```python
def compare_with_file(self, tokens_input):
    # Selección de archivo...
    # Análisis del archivo seleccionado...
    # Comparación de tokens...
```

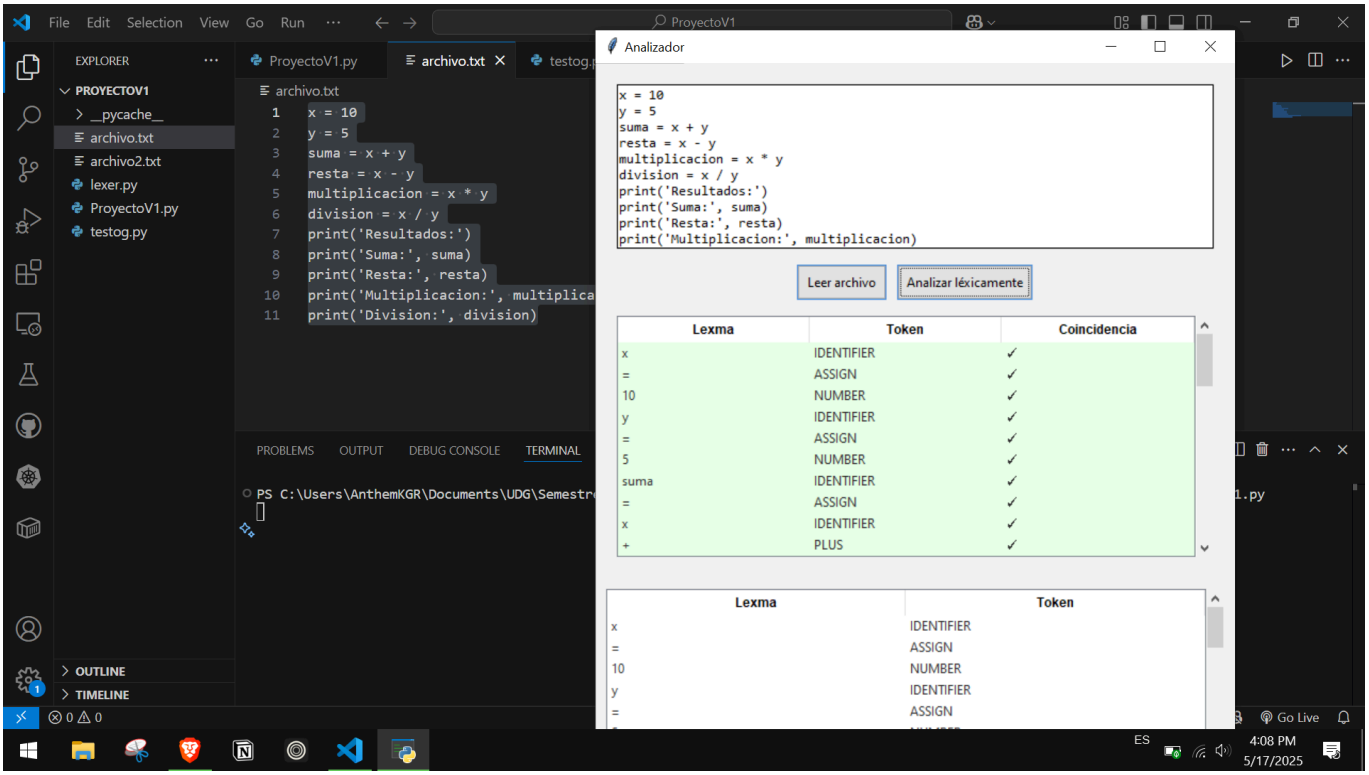
Verifica si el script se está ejecutando directamente
Se crea la ventana principal de tkinter
Se Inicializa la aplicación
Inicia el bucle principal de eventos de la interfaz gráfica

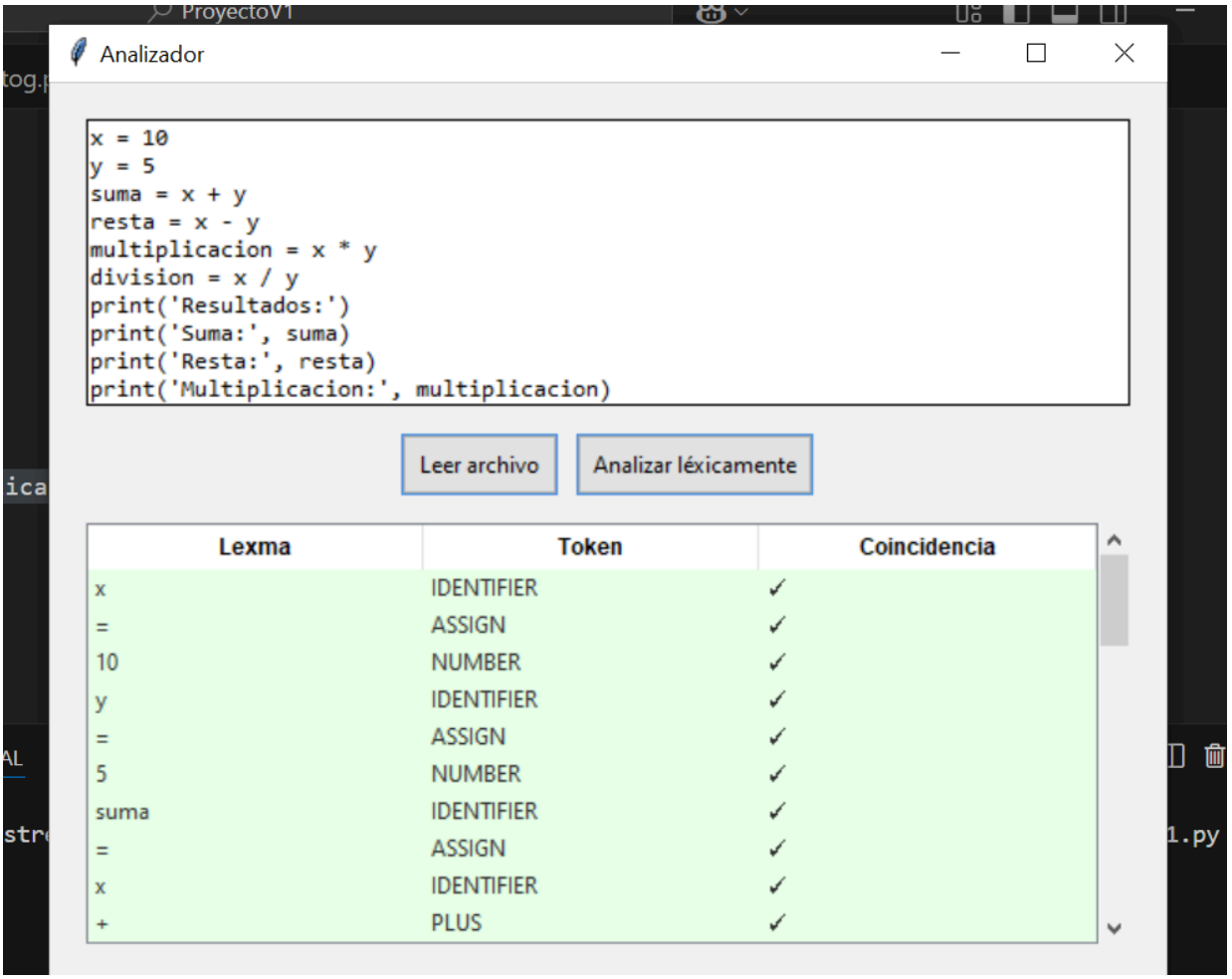
```python
if __name__ == "__main__":
    root = tk.Tk()
    app = LexicalAnalyzerGUI(root)
    root.mainloop()
```

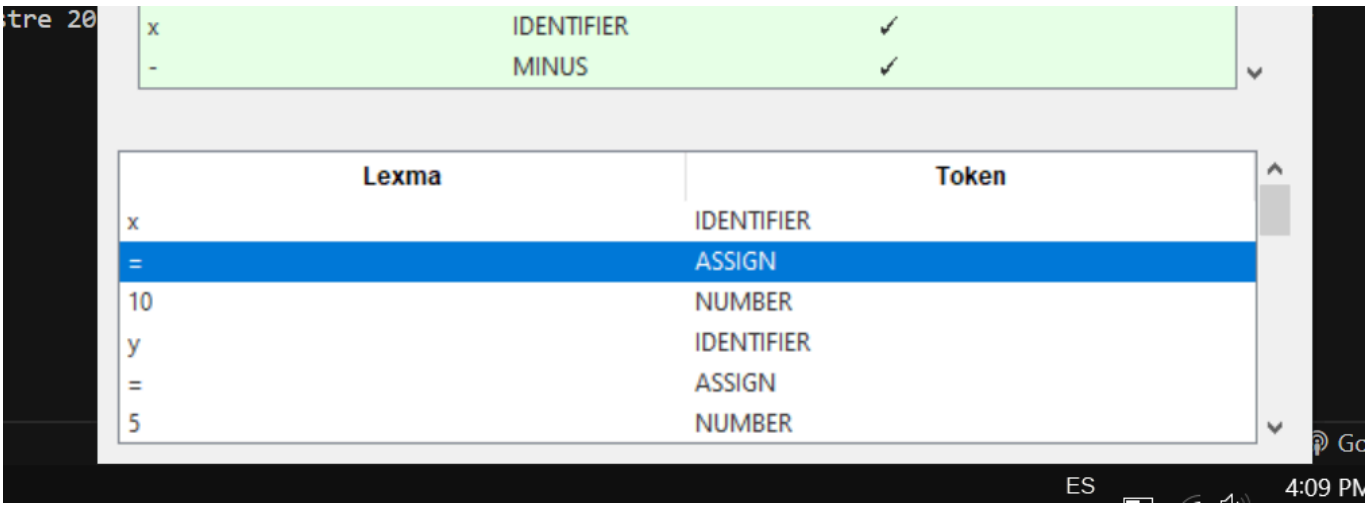
Capturas de pantalla






File Edit Selection View Go Run
ProyectoV1
EXPLORER
PROYECTOV1
__pycache__
archivo.txt
archivo2.txt
lexer.py
ProyectoV1.py
testog.py
ProyectoV1.py
archivo.txt
testog.p
archivo.txt
1 x = 10
2 y = 5
3 suma = x + y
4 resta = x - y
5 multiplicacion = x * y
6 division = x / y
7 print('Resultados:')
8 print('Suma:', suma)
9 print('Resta:', resta)
10 print('Multiplicacion:', multiplica
11 print('Division:', division)
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\AnthemKGR\Documents\UDG\Semestr
OUTLINE
TIMELINE
Analizador
x = 10
y = 5
suma = x + y
resta = x - y
multiplicacion = x * y
division = x / y
print('Resultados:')
print('Suma:', suma)
print('Resta:', resta)
print('Multiplicacion:', multiplicacion)
Leer archivo
Analizar léxicamente
Lexma Token Coincidencia
x IDENTIFIER ✓
= ASSIGN ✓
10 NUMBER ✓
y IDENTIFIER ✓
= ASSIGN ✓
5 NUMBER ✓
suma IDENTIFIER ✓
= ASSIGN ✓
x IDENTIFIER ✓
+ PLUS ✓
Lexma Token
x IDENTIFIER
= ASSIGN
10 NUMBER
y IDENTIFIER
= ASSIGN
ES
4:08 PM
5/17/2025


ProyectoV1
Analizador
x = 10
y = 5
suma = x + y
resta = x - y
multiplicacion = x * y
division = x / y
print('Resultados:')
print('Suma:', suma)
print('Resta:', resta)
print('Multiplicacion:', multiplicacion)
Leer archivo
Analizar léxicamente
Lexma Token Coincidencia
x IDENTIFIER ✓
= ASSIGN ✓
10 NUMBER ✓
y IDENTIFIER ✓
= ASSIGN ✓
5 NUMBER ✓
suma IDENTIFIER ✓
= ASSIGN ✓
x IDENTIFIER ✓
+ PLUS ✓



Conclusión:

DIEGO ANDRES HERNANDEZ RODRIGUEZ

Juan Antonio Pérez Juárez:
Uffffff, afortunadamente ya tenía muchos elementos de la práctica anterior, lo que más me quebré la cabeza por el diseño en tkinter, que afortunadamente hace 6 meses terminé un curso de python donde se enfocaba mucho en tkinter, así que a pesar de que fue difícil hacer las conexiones con las funciones como parser(), fue medianamente sencillo y con ayuda de la IA para los colores, me ayudó a no demorar mucho.
Considero que este tipo de prácticas, a corto plazo no les encuentro valor, pero supongo que adquieren valor con el paso o desarrollo de la carrera, por que a pesar de que conozco mejor el cómo funcionan los compiladores.
No me veo capaz de poder hacer un compilador desde 0 o como dicen en ingles From scratch to Compiler, quiza algunas cositas pero incluso el ASM se me complica muchísimo así que no creo dedicarme a esta rama de la programación.

Pero el tener los conocimientos me hace estar por encima de la media de los programadores y eso es una herramienta invaluable.

Andre Maximiliano Calderon Acero