Universidad De San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas

Lenguajes formales y de programación
Sección “A-”

“MANUAL TÉCNICO”

Samuel Alejandro Pajoc Raymundo.

Carné: 201800665

Objetivos

General:

Brindar al lector una guía que contenga la información del manejo de clases, atributos, métodos y del desarrollo de los menús, así mismo el desarrollo del autómata implementado para analizar el texto cargado al área correspondiente, el propósito de esto es facilitar futuras actualizaciones o futuras modificaciones realizadas por terceros.

Específicos:

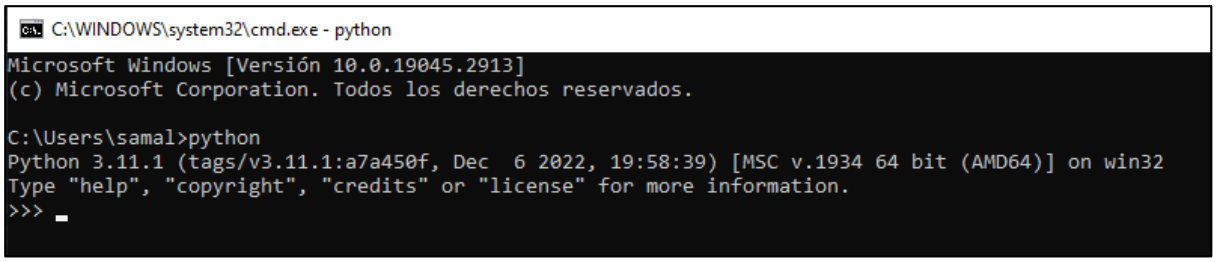
- Mostrar al lector una descripción lo más completa y detallada posible del SO, IDE entre otros utilizados para el desarrollo de la aplicación.

- Proporcionar al lector una explicación técnica/formal de los procesos y relaciones entre clases, métodos y atributos que conforman la parte operativa de la aplicación.

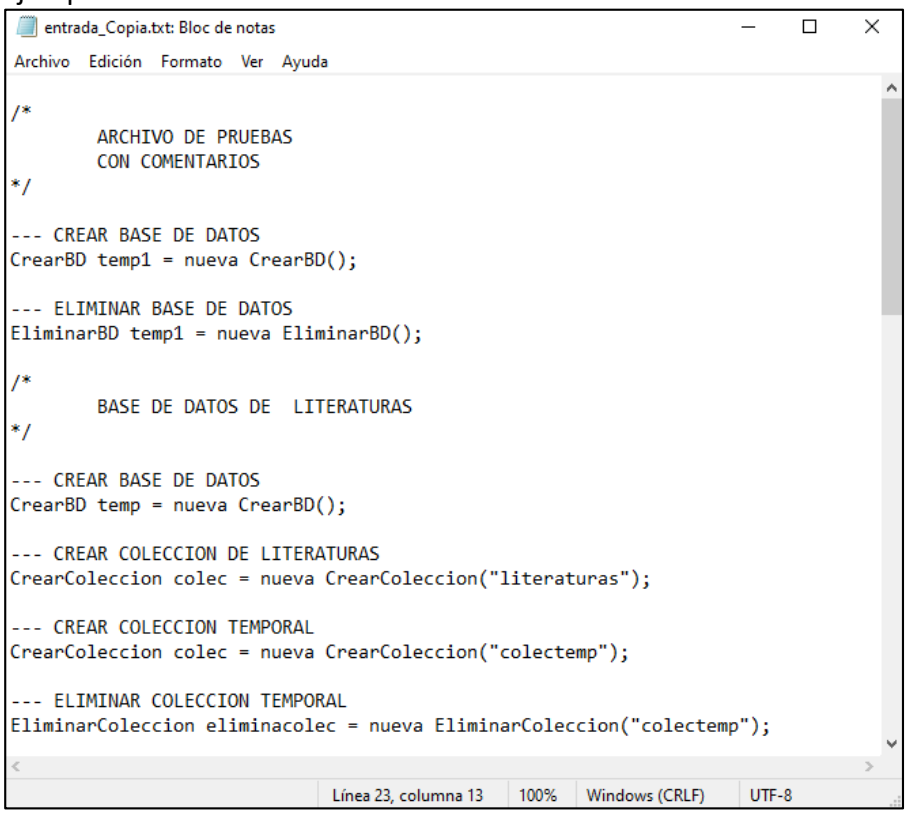
Introducción

Este manual técnico tiene como finalidad dar a conocer al lector, una idea base para que pueda realizar futuras actualizaciones al software, indicando el IDE utilizado para su creación, su versión, requerimientos del sistema, etc.

La aplicación tiene como objetivo analizar el contenido de un archivo de texto plano, en este caso se utilizara el formato (.txt), inicialmente cargado por el usuario en su correspondiente menú, además, proporcionar de una manera más amigable al usuario la detección de errores y tokens presentes en dicho archivo; la aplicación fue creada con el lenguaje de python Version 3.11.1 (64-bits).



Ejemplo del archivo de entrada:



El formato del archivo es:

‘ / ‘ : Diagonal, indica el inicio o fin de un comentario multi-línea, este debe ir acompañado de un *.

‘ * ‘ : Asterisco, forma parte de la indicación de un inicio o fin de un comentario multi-línea.

[a-z][A-Z] : letras de la A - Z, mayúsculas y minúsculas. Permitiendo formar palabras.

[0-9] : Números del 0 al 9. Permitiendo formar números enteros. Únicamente se permiten si forma parte del nombre de una variable de una función o se encuentra dentro de un campo string “texto”.

‘ . ‘ : Punto, Siendo parte únicamente de la formación de oraciones.

{ , } : Llaves, indican el inicio o cierre de un JSON.

‘ = ‘ : Igual, indica la continuación y la creación de una instrucción.

(,) : Paréntesis, indica el inicio o cierre de un parámetro que acompaña a una instrucción.

‘ ; ’ : Punto y coma, indica el fin de una instrucción.

“ “ : comillas dobles, indican el inicio o fin de un campo, valor o JSON de una instrucción.

‘:’ : Dos puntos, indican el inicio del valor que acompaña al campo llave de un JSON.

‘ – ‘ : Guion, indicando el inicio de un comentario de una sola línea.

$: Dólar, indica el inicio de un campo llave-valor que sustituirá a otro valor.

‘ , ‘ : Coma, indica que un JSON contendrá más campos llave-valor.

- Si este formato no se respeta, el programa dará un mensaje de error y no se traducirán las instrucciones (únicamente lo hará si el archivo de entrada no cuenta con -ningún error-), dado que el compilamiento no estaría completo. Por lo que se habilitará el botón de “Ver errores” y podrá ver donde se detectó el fallo, esto lo podrá ver el usuario dado que se encuentra dentro del menú “Menu archivo”.

- Además, si el contenido del archivo posee caracteres no permitidos, el programa indicara su ubicación (columna, fila), de tal forma que se pueda proceder a ser removido o sustituido por uno valido, esto lo podrá observar el usuario en el botón “Ver errores”, dado que esta opción se encuentra dentro del menú “Menu archivo”.

Descripción de la Solución

Esta aplicación fue desarrollada aplicando diversos paradigmas de programación como programación orientada a objetos (POO) y programación estructural.

- Se desarrolló distintos tipos de clases para poder tener un mejor control del análisis del programa o de los mismos datos obtenidos en el archivo cargado. Las clases que se crearon fueron:

    - **analizarTexto.py :** Esta clase contiene el autómata finito determinista, el cual permite analizar carácter por carácter, el cual es el análisis léxico, y así obtener los datos más esenciales del archivo previamente cargado, además allí mismo se realiza la implementación del análisis sintáctico. Al finalizar se retornan los datos más esenciales para realizar los grafos y la traducción de instrucciones.

    - se hace un llamado a travez de un objeto creado en el menú principal llamado "analizarT". Un ejemplo de esta clase es "analizarT.AnalizarEntrada(texto)".

    - **escrituraComandos.py :** esta clase permite escribir las instrucciones correspondientes, esto se realiza de acuerdo al tipo de instrucción que se recibe, las instrucciones se almacenan en una variable llamada "texto", el cual se retorna para luego ser implementado en otra clase, la cual permite realizar la escritura de un archivo nuevo el cual únicamente contiene las instrucciones traducidas, y esta variable a su vez se utiliza para mostrar el texto contenido en el segundo espacio de texto, dentro del "menu archivo".

    - **Main.py :** esta clase contiene todos los elementos que permiten crear el entorno grafico del programa, además contiene sub-métodos los cuales le permiten un funcionamiento correcto a todos los botones y entradas de texto ubicado en cada ventana desplegable.

- **crearGrafos.py :** Esta clase permite la elaboración del grafo de errores y tokens (desarrollado en graphviz, pero a su vez implementando un ambiente lógico por medio del lenguaje de python). Esta clase también cuenta con una función definida, la cual escribe un archivo en formato dot. Y luego solicita abrirlo, para que de este modo, el usuario pueda visualizar su resultado en una imagen con formato png, el archivo se encuentra ubicado en la carpeta llamada grafos (dentro de la carpeta del proyecto), el nombre del archivo se contiene el formato: "NombreArchivo.dot.png".

  - grafo(self, datos)
    - →Esta función aplica para el grafo de los errores o tokens.
  - esta función recibe como paramentos los valores obtenidos por el analizador sintáctico y léxico, este grafo se finaliza en formato dot.png.

  Para poder almacenar el archivo en formato dot por medio de python, se utilizó la librería "render", la cual permite crear la imagen png a travez del archivo escrito dot.

  Para poder abrir el archivo (imagen en formato png), se utilizó la función "webbrowser" y se colocó un alias de la misma como "wb", y por medio de su función "wb.open_new(>>grafos/nombre_archivo.dot.png<<)", es que se logra desplegar la imagen generada.

- Cada vez que se realice una acción importante o ocurra un problema, el programa le indicara el estado del programa, por medio de pequeños mensajes desplegados en ventanillas llamadas "mesagebox", se implementó dos tipos de advertencia para mostrar los estados, los cuales son :
  - .showerror
  - .showinfo

- Si el archivo de entrada cuenta con un error; no se realizara la interpretación de instrucciones, únicamente se procederá hasta que el error o errores hayan sido corregidos, estos erroes se pueden observar en la opción "Ver errores", la cual únicamente se habilitara cuando se detecten errores.

- Una vez se logre realizar la interpretación de instrucciones, además de ser mostradas en el área correspondiente, estas serán escritas en un archivo de texto plan llamado "comandos_MongoDB.txt", este se ubicará en la carpeta "traduccion".

Tokens

- **Expresión Regular (Idea principal):**

/*LETRA+*/ | ---LETRA+ |
LETRA+\sLETRA+\s=\sLETRA+\sLETRA+\([|"LETRA+"[|,"{[["LETRA+":"LETRA+",?]+|"LETRA+":"LETRA+"},{$LETRA+:{"LETRA+":"LETRA+"}]}"]]\);

- **Expresión Regular (aplicada):**

/*(\w)+*/ | ---(\w)+ | \w+\s\w+\s=\s\w+\s\w+\(([|"\w+"(|,"\{(("\w+":"\w+",?)+|"\w+":"\w+"\},\{\$\w+:\{"\w+":"\w+"\})\}"))\);

- Ejemplo de visualización de tokens por medio de función “Ver tokens”:

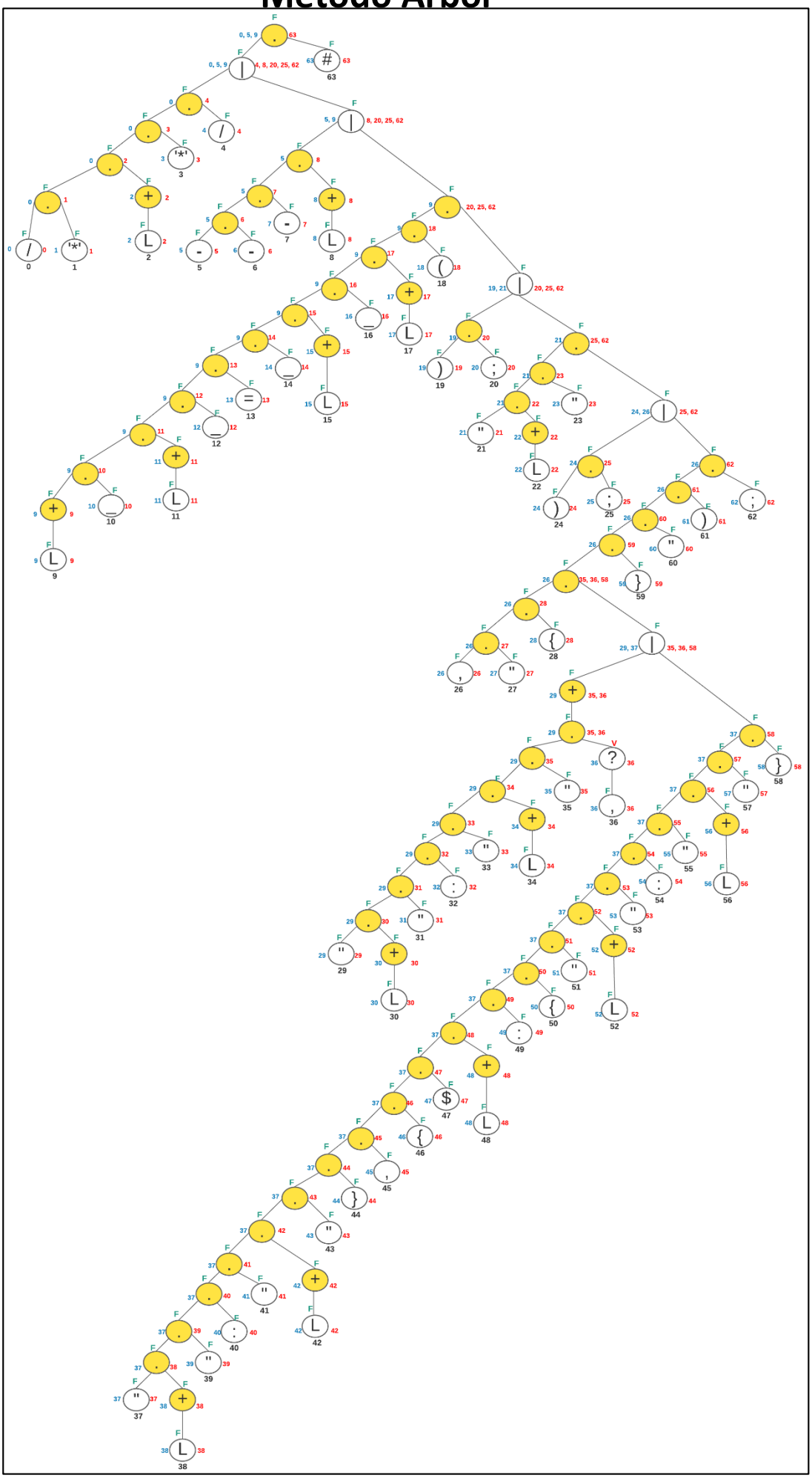
No.	TOKEN	LEXEMA
1	Apertura_ComentMulti	/*
2	ComentMulti	ARCHIVO DE PRUEBAS CON COMENTARIOS
3	Cierre_ComentMulti	*/
4	ComentUnaLinea	---
5	ContenidoComentUnaLinea	CREAR BASE DE DATOS
6	FuncionValida	CrearBD
7	Nom_VariableComando	temp1
8	Signo_Igual	=
9	Afirmacion_FuncionNueva	nueva
10	2daV_FuncionValida	crearbd
11	Parentesis_Apertura	(
12	Parentesis_Cierre	)
13	FinInstruccion	;

Lista Tokens Validos:

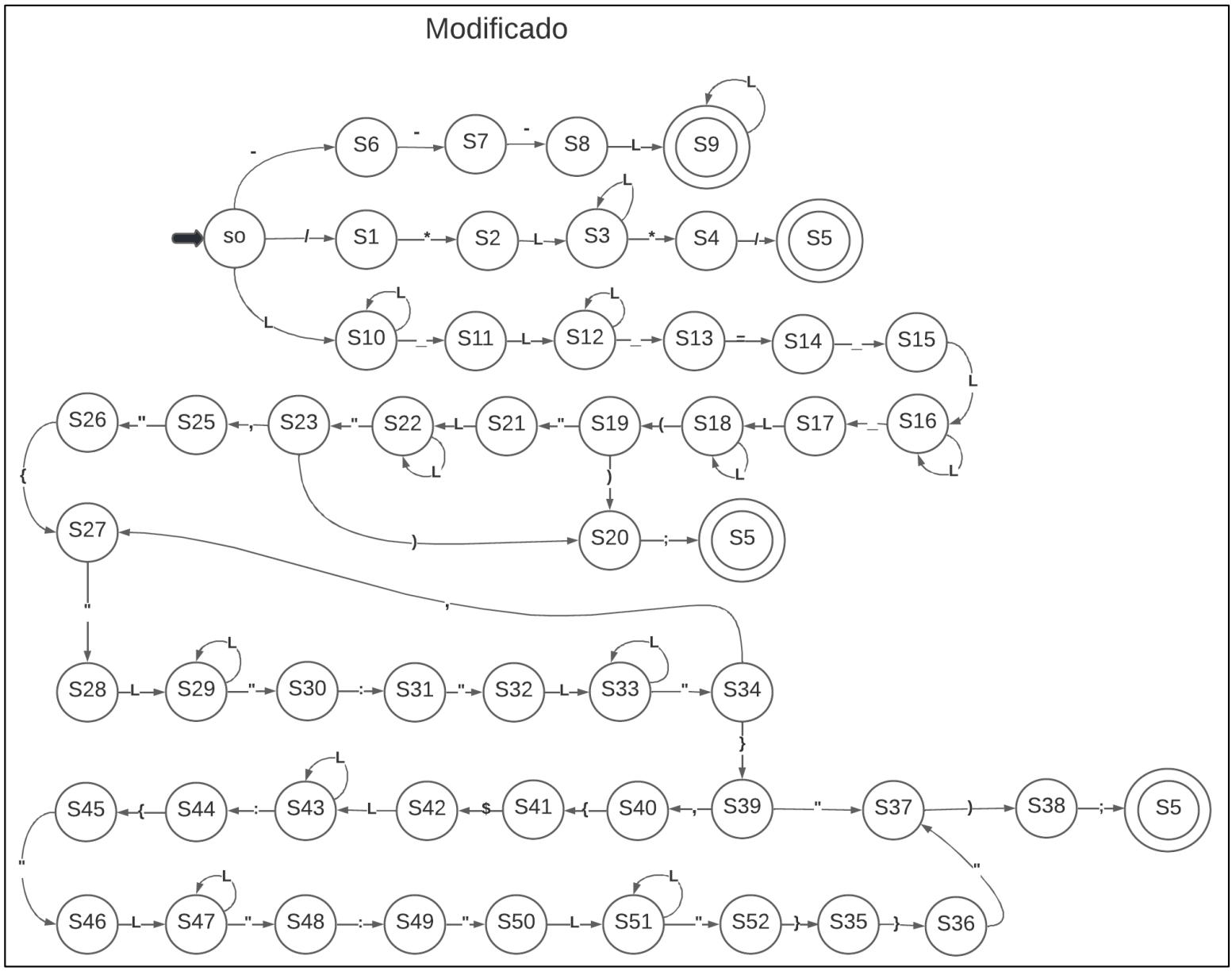
No.	TOKEN	Patrón (Expresión Regular)
1	Apertura_ComentMulti	/*
2	ComentMulti	([a-z][A-Z] | [0-9])+
3	Cierre_ComentMulti	*/
4	ComentUnaLinea	---
5	ContenidoComentUnaLinea	([a-z][A-Z] | [0-9])+

6	FuncionValida	([a-z]|[A-Z])+
7	Nom_VariableComando	[a-z]|[A-Z] ([a-z]|[A-Z]|[0-9])+
8	Signo_Igual	=
9	Afirmacion_FuncionNueva	([a-z]|[A-Z])+
10	2daV_FuncionValida	([a-z]|[A-Z])+
11	Parentesis_Apertura	(
12	Parentesis_Cierre	)
13	FinInstruccion	;
14	ComillasDob_Apertura	“
15	NombeColeccion	([a-z]|[A-Z])+
16	ComillasDob_Cierre	“
17	MasParametros	,
18	Llave_Apertura	{
19	Campo_Llave	([a-z]|[A-Z])+
20	Dospuntos	:
21	Campo_Valor	([a-z]|[A-Z])+
22	Llave_Cierre	}
23	ActualizarInfo	$
24	Campo_Establecer	([a-z]|[A-Z])+

Método Árbol



AFD (Autómata Finito Determinista)



Editados:

->S34 se desvinculó de S35, ya que no tenian sentido.

->S34 se vinculó con S27, ya que ese estado cumple con la continuidad de la verificación.

ELIMINADOS:

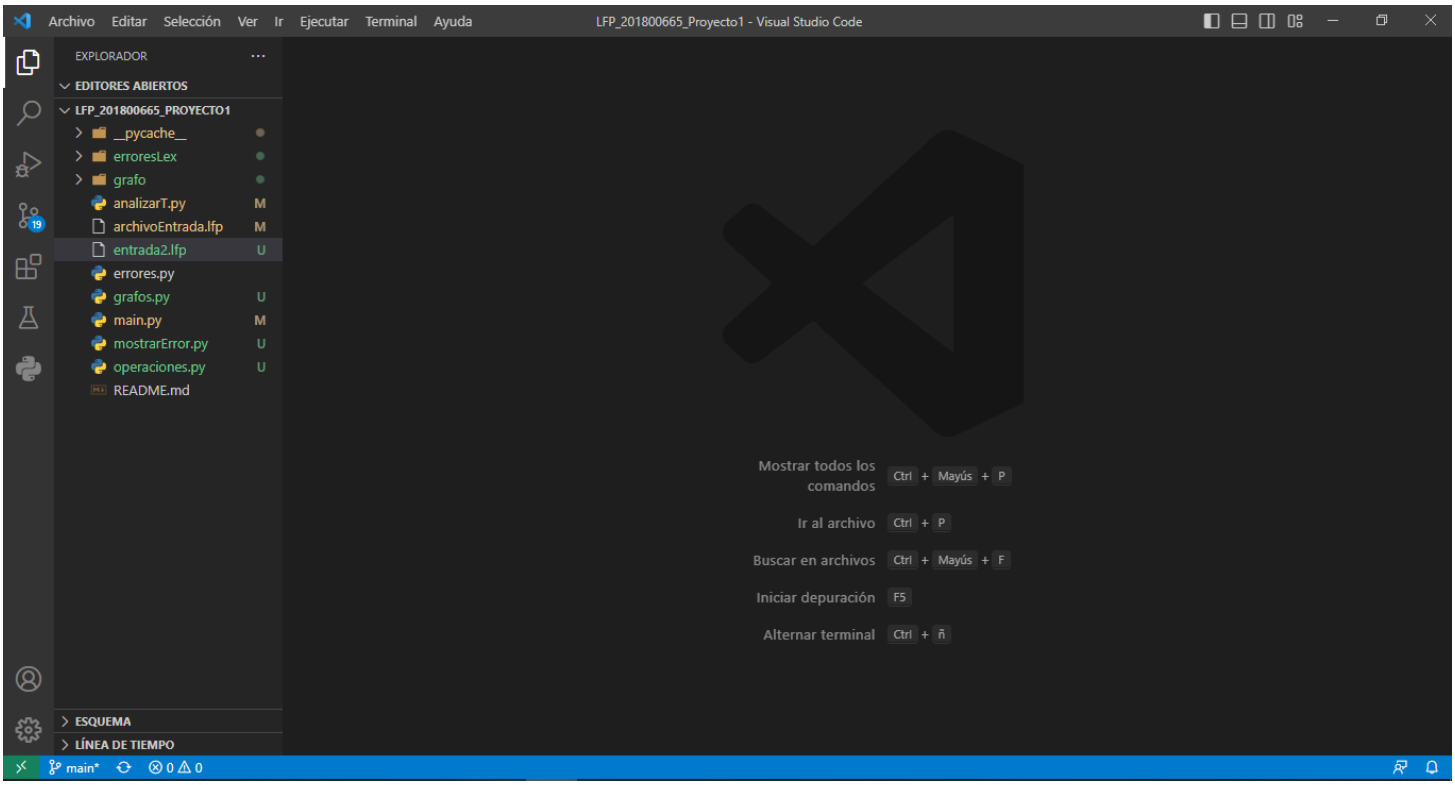
->S24(Hace lo mismo que S20)

-> S53(y los nodos que lo acompañen, No tiene sentido)

->Se eliminó la conección que habie entre S28 a S35, ya que no tenía sentido que los uniera una (,).

IDE

El IDE con el que se desarrolló el proyecto fue Visual Studio Code.



Librerías Utilizadas

Las librerías utilizadas para el desarrollo de este proyecto fueron:

```
import tkinter
import tkinter.font as tkFont
from tkinter import DISABLED, END, NORMAL, messagebox
import webbrowser as wb
import easygui
from graphviz import render
```