Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ciencias y Sistemas

Lenguajes Formales y de Programación

Catedrática:

Inga. Asunción Mariana Sic Sor

Tutor académico:

Enrique Alejandro Pinula Quiñonez

“MANUAL TECNICO PROYECTO 2”

Melvin Eduardo Ordoñez Sapón RA | 3256215860802

Guatemala, junio del 2024

“Id y enseñad a todos”

INDICE

Contenido

ACERCA DEL PROGRAMA:

Editor Utilizado: Visual Studio Code

Versión de Python: Python 3.12.3

Programado en Sistema Operativo: Windows 11 Pro

Terminal Utilizada: Git Bash

Lenguaje de Programación Utilizado: Python

Compatibilidad: Windows, Linux, macOS

Librerías Utilizadas: Tkinter, Graphviz, Pandas

Versión Del Programa: 1.0

LOGICA DE LA SOLUCION

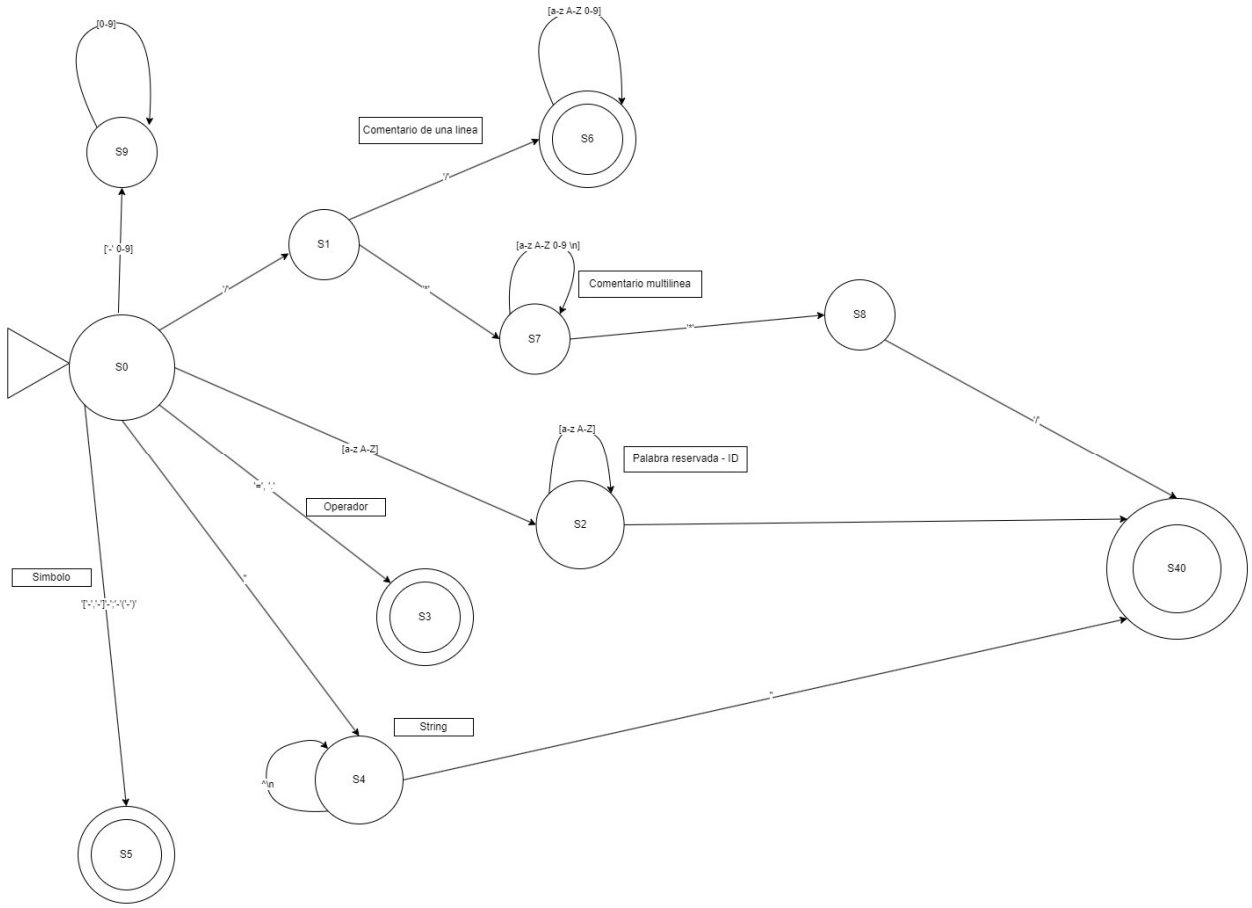
La lógica para la solución del programa que almacena lista de datos en un archivo csv consiste en los siguiente:

1. Análisis léxico
2. Almacenamiento de cada token encontrado
3. Análisis sintáctico
4. Aplicación de Reglas
5. Almacenar las instrucciones
6. Realizar las acciones

ANALISIS LEXICO

En la parte del análisis léxico el programa se basa en una serie de estados en donde cada uno representa un carácter valido dentro de nuestro lenguaje especificado en el archivo de entrada y así poder identificar cuáles son los tokens válidos e identificar los errores léxicos.

Para el AFD se utilizaron estados de transición y estados de aceptación en donde se toma como valido el lexema formado, el Autómata que describe el comportamiento de los estados es el siguiente:


[0-9]
S9
[a-z A-Z 0-9]
S6
Comentario de una linea
'/'
S1
[-' 0-9]
'/'
'*'
[a-z A-Z 0-9 \n]
Comentario multilinea
S7
'*'
S8
S0
'/'
[a-z A-Z]
[a-z A-Z]
Palabra reservada - ID
S2
'=', '.'
Operador
S40
Simbolo
'[-','-']','.',':-'('-')'
S3
'
String
^\n
S4
"
S5

ALMACENAMIENTO DE CADA TOKEN

Para el almacenamiento de los tokens el lexer se encarga de ir almacenando en un arreglo que tokens son válidos, al momento de reconocer un token como valido se guarda y el estado regresa al inicio para analizar el carácter siguiente en la cola del archivo de entrada.

ANALISIS SINTACTICO

En el análisis sintáctico se verifica de que cada token coincida con las reglas gramaticales que es la siguiente

-----Terminales-----

tk_palabraArray(PalabraReservada)

tk_id(Identificador)

tk_igual(Operador)

tk_new(PalabraReservada)

tk_corcheteApertura(Simbolo)

tk_coma(Simbolo)

tk_corcheteCierre(Simbolo)

tk_puntoComa(Simbolo)

tk_numero(Numero)

tk_punto(Operador)

tk_sort(PalabraReservada)

tk_parentesisApertura(Simbolo)

tk_asc(PalabraReservada)

tk_true(PalabraReservada)

tk_false(PalabraReservada)

tk_parentesisCierre(Simbolo)

tk_save(PalabraReservada)

tk_string(Cadena)

-----Producciones-----

<inicio> ::= <instrucciones>

<instrucciones> ::= <instruccion> <instrucciones>

| epsilon

<instruccion> ::= <declaracion>

| <instruccionID>

<instruccionID> ::= tk_id(Identificador) tk_puntoComa(Simbolo)
<accionArreglo>

<accionArreglo> ::= <ordenamiento>

| <guardar>

<declaracion> ::= tk_palabraArray(PalabraReservada) tk_id(Identificador) tk_igual(Operador) tk_new(PalabraReservada) tk_palabraArray(PalabraReservada) tk_corcheteApertura(Simbolo) <listaElementos> tk_corcheteCierre(Simbolo) tk_puntoComa(Simbolo)

<listaElementos> ::= <elemento> tk_puntoComa(Simbolo) <listaElementos>

| <elemento>

| epsilon

<elemento> ::= tk_numero(Numero)

| tk_string(Cadena)

<ordenamiento> ::= tk_sort(PalabraReservada) tk_parentesisApertura(Simbolo) tk_asc(PalabraReservada) tk_igual(Operador) tk_true(PalabraReservada) tk_parentesisCierre(Simbolo) tk_puntoComa(Simbolo)

| tk_sort(PalabraReservada) tk_parentesisApertura(Simbolo) tk_asc(PalabraReservada) tk_igual(Operador) tk_false(PalabraReservada) tk_parentesisCierre(Simbolo) tk_puntoComa(Simbolo)

<guardar> ::= tk_save(PalabraReservada) tk_parentesisApertura(Simbolo) tk_string(Cadena) tk_parentesisCierre(Simbolo) tk_puntoComa(Simbolo)

-----No_Terminales----

<inicio>

<instrucciones>

<instruccion>

<instruccionID>

<accionArreglo>

<declaracion>

<listaElementos>

<elemento>

<ordenamiento>

<guardar>

APLICACIÓN DE REGLAS

En esta parte se aplican las reglas de producción a la secuencia de tokens generados

ALMACENAR LAS INSTRUCCIONES

A medida que se van reconociendo las producciones en el análisis sintáctico estos se guardan para después realizar todo lo especificado en el archivo de entrada y poder crear instancias de listas, ordenarlas y exportarlas a un archivo csv.

REALIZAR ACCIONES

Luego de terminar con el análisis sintáctico se procede a realizar las acciones que fueron definidas en el lenguaje de entrada y así conseguir todo lo especificado.

METODO IMPORTANTES

analizar:

Este método recibe el texto que se le pasa al programa a través de un archivo de entrada y analiza cada carácter del texto recibido para irlos clasificando como tokens validos o errores léxicos (Método en la clase Lexer).

parser:

Este método es el punto de entrada principal para el análisis sintáctico de una secuencia de token antes definidas por el lexer. Este método se encarga de coordinar la llamada a las reglas de producción definidas en la gramática del lenguaje a analizar.

realizar_acciones:

Con este método se verifica que no haya un error sintáctico para proseguir, luego de eso se llama al método de **ordear_elementos** que es el encargado de ordenar todas las listas que fueron indicadas en la entrada. También se ejecuta el método **exportar_csv** que traslada la información que tienen las lista a un archivo csv.