

**Universidad de San Carlos de Guatemala**  
**Facultad de Ingeniería**  
**Escuela de Ciencias y Sistemas**  
**Laboratorio de Lenguajes Formales y de Programación**

# **Manual Técnico**

Proyecto 1: Analizador léxico

**Catedrático:** Mario Josué Solís Solórzano

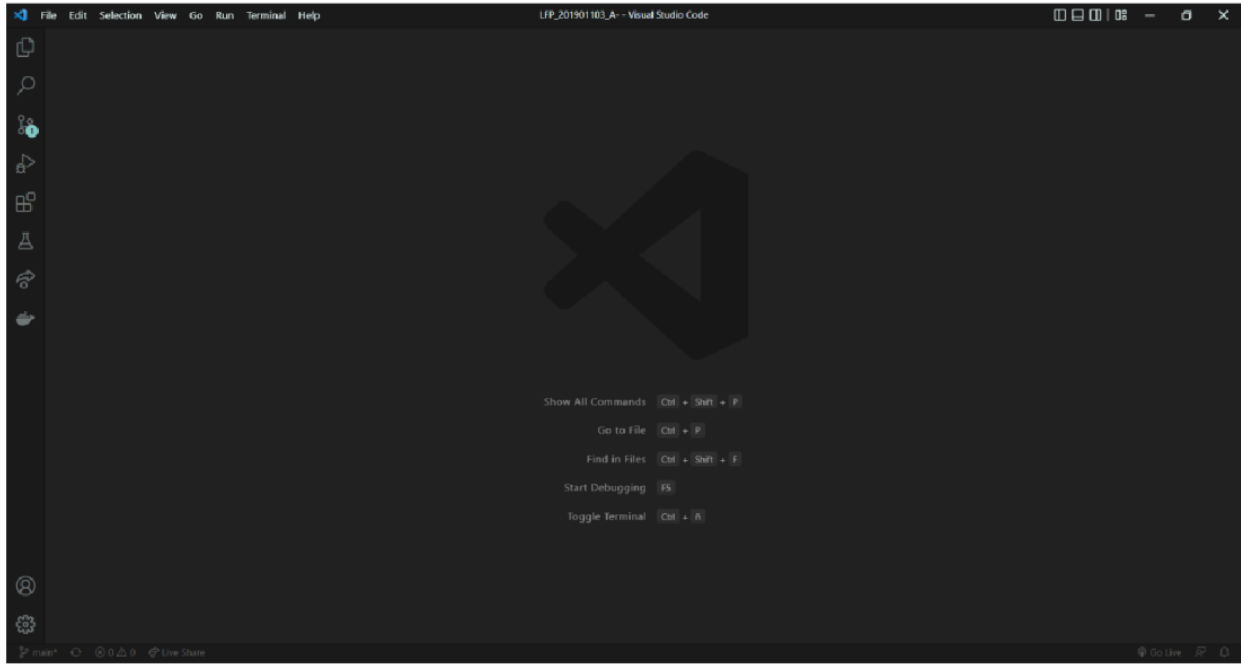
**Sección:** A-

**Nombre:** Josué Daniel Rojché García

**Carné:** 201901103

## Requerimientos del Sistema

En la realización del software el Editor de texto que se utilizó fue Visual Studio Code.



El programa fue realizado con la versión de Python 3.11.1 para sistemas operativos de 64 bits.

Python 3.11.1

## Analizador Léxico

Para que el programa funcione correctamente se realizaron diversas validaciones y manejo de errores para que no finalice indebidamente durante la ejecución de esta. Por lo cual explicaremos lo que realiza el código.

Primero se importan las librerías necesarias, las cuales se utilizarán más adelante, por ejemplo: la librería tkinter, la cual servirá para las ventanas graficas

```
1  import os
2  import tkinter as tk
3  from tkinter import CENTER, RIGHT, Y, Scrollbar, filedialog, Tk, ttk
4  from tkinter.messagebox import showerror, showinfo, showwarning
5  import webbrowser
6  from automataAFD import AFD
```

### Menú Principal

Para mostrar el menú principal se crea el main, el cual será el que se ejecute inicialmente, y comenzará por mostrar la pantalla principal.

La pantalla se crea con tk.Tk(), con title se le agrega un titulo a la pestaña, con geometry indicamos el tamaño que tendrá dicha ventana indicando el ancho por altura, con configure se indica el color de fondo que tendrá la ventana, resizable sirve para indicar que la ventana sea de tamaño fijo y que el usuario no pueda agrandarla o hacerla más pequeña.

Para crear una barra de menú se utiliza tk.Menu(), el cual sirve para agregarle los datos que sean necesarios, en este caso se crea el espacio para el manejo de los archivos por lo que se utiliza add\_cascade() con sus respectivos parámetros, el primer parámetro indica que se agregará al menú de barra, y el label le indica el nombre que tendrá para esas opciones, para agregar las subopciones se utiliza add\_command() el cual lleva como parámetros los siguientes, label el cual indica el nombre de la opción a utilizar, luego command al cual se le asigna el método al que accederá para realizar la acción que corresponda, luego con state se indica que la opción no estará disponible si no se carga un archivo con anterioridad.

Con tk.Text() se agrega dos áreas de texto que servirán para editar el archivo y mostrar los errores del archivo al momento de realizar las opciones que correspondan a partir que se cargue el archivo en el sistema, con configure() se le indica el color de fondo que tendrá con bg, y state para indicar que al principio no se pueda editar en el área de texto.

Con tk.label() se crea un texto en una línea que estará plasmado para indicar cual cuadro de texto es para el archivo cargado y el que muestra los errores.

Con config() se agrega el parámetro que agrega a la ventana la barra de menú y con mainloop() se indica que que la ventana será visible para el usuario.

```

102 if __name__ == '__main__':
103     menu = tk.Tk()
104     menu.title("PROYECTO NO.1")
105     menu.geometry("900x400")
106     menu.configure(bg="#212F3C")
107     menu.resizable(False, False)
108     #Crea la barra del menu
109     barra_Menu = tk.Menu()
110     #Crea el primer elemento el cual será para archivo y se enlaza a la barra
111     menuArchivo = tk.Menu(barra_Menu, tearoff=False)
112     #Al add_command se le puede pasar el parametro accelerator="Ctrl+N" para agregar un atajo con el teclado
113     menuArchivo.add_command(label="Abrir", accelerator="Ctrl+N", command= abrir)
114     menu.bind_all("<Control-n>", abrir )
115     menuArchivo.add_command(label="Guardar", command= guardar, state=tk.DISABLED)
116     menuArchivo.add_command(label="Guardar Como", command=guardarComo, state=tk.DISABLED)
117     menuArchivo.add_command(label="Analizar", command= analizar, state=tk.DISABLED)
118     menuArchivo.add_command(label="Errores", command= errores, state=tk.DISABLED)
119     menuArchivo.add_separator()
120     menuArchivo.add_command(label="Salir", command=menu.quit, activebackground="Red")
121
122     menuAyuda= tk.Menu(barra_Menu, tearoff=False)
123     menuAyuda.add_command(label="Manual de Usuario", command= manualUsuario)
124     menuAyuda.add_command(label="Manual Tecnico", command= manualTecnico)
125     menuAyuda.add_command(label="Temas de Ayuda", command= temasAyuda)
126
127     # a la barra menú le agregamos el menuArchivo
128     barra_Menu.add_cascade(menu= menuArchivo, label= "Archivo")
129     barra_Menu.add_cascade(menu= menuAyuda, label= "Ayuda")
130
131     textLeer = tk.Text()
132     textLeer.configure(bg="#C8C885", state="disabled")
133     textLeer.place(x= 5, y =5, height= 350, width= 440)
134
135     textErrores = tk.Text()
136     textErrores.configure(bg="#BFBF02", state="disabled") #, state="disabled"
137     textErrores.place(x= 455, y =5, height= 350, width= 440)
138
139     label1 = tk.Label(menu, text="Archivo abierto", bg="#212F3C", fg="FFFFFF",width= 20, font=("Arial", 13)).place(x= 125, y =370)
140
141     label1 = tk.Label(menu, text="Archivo Errores", bg="#212F3C", fg="FFFFFF",width= 20, font=("Arial", 13)).place(x= 575, y =370)
142
143     menu.config(menu=barra_Menu)
144     menu.mainloop() # Permite mostrar la ventana

```

## Método abrir

Primero se crea la variable almacenar la cual servirá para guardar todo el texto que tiene el archivo a abrir, la variable url guardará la dirección donde se encuentra dicho archivo, y para manejar el analizador realizamos la instancia hacia la clase AFD().

Ahora en el método abrir se utiliza `filedialog.askopenfilename()` para abrir una ventana que permite acceder a los archivos y seleccionar el que se requiera utilizar en el programa, el cual debe ser con la extensión `.json`, luego con `if` se valida si se ha seleccionado algún archivo, si el archivo es cargado entonces abre el archivo con el método `open`, y con `read()` lee el contenido del mismo y lo almacena en la variable creada globalmente, además se agregan estos datos al cuadro de texto que se creó inicialmente para poder editar el contenido del mismo, y se habilitan las demás opciones de la barra

de menú con entryconfig, y con showinfo indicamos un mensaje para dar a entender al usuario que el archivo se cargó en el sistema de forma exitosa.

```
9  almacenar = ""
10 urlAlmacenar = ""
11 enviandoAnálisis = AFD()
12 #Metodos y Funciones para la sección de Archivo *****
13 def abrir(event = None):
14     global urlAlmacenar
15     urlArchivo = filedialog.askopenfilename(initialdir=".", title="Seleccione un Archivo", filetypes=(("Archivo json", "*.json"), ("all files", "*.*")))
16     if urlArchivo != "":
17
18         leer = open(urlArchivo, "rt")
19         urlAlmacenar = urlArchivo
20         global almacenar
21         almacenar = leer.read()
22         leer.close()
23
24         textLeer.configure(state=tk.NORMAL)
25         textLeer.insert("1.0", almacenar)
26
27         menuArchivo.entryconfig(1,state = tk.NORMAL)
28         menuArchivo.entryconfig(2,state = tk.NORMAL)
29         menuArchivo.entryconfig(3,state = tk.NORMAL)
30         menuArchivo.entryconfig(4,state = tk.NORMAL)
31
32         showinfo(title="Abierto", message="Archivo leído exitosamente")
33     else :
34         #showerror(title="Error", message="El tamaño máximo de organismos es: 1000 \nPorfavor ingrese menos organismos")
35         showwarning(title="Advertencia", message="No ingresó ningún archivo")
```

### Método guardar

Se crea una variable que tendrá el método open para acceder al archivo que se abrió y poder editar en él, guardando lo que obtenga del área de texto con el método get, y este se escribe con write en el archivo, luego se cierra este para evitar el seguir usándolo sin ser necesario.

```
37 def guardar(event = None):
38     saveArchivo = open(urlAlmacenar, "w")
39     saveArchivo.write(textLeer.get("1.0","end"))
40     saveArchivo.close()
```

### Método guardar como

Se crea el método para permitir guardar lo que se editó en el cuadro de texto con otro nombre y en la ubicación que el usuario requiera, para ello se comienza creando la variable con filedialog.asksaveasfilename() el cual permite abrir la ventana para seleccionar la ruta donde se guardará el archivo y agregarle en nombre al mismo.

Se valida con if si el usuario ha ingresado alguna ruta y agregado un nombre al archivo, si el usuario ingresa correctamente los datos se utiliza open para crear el archivo y permitir escribir en él con el método write() y este obtiene con get los datos que se encuentran en el cuadro de texto y para finalizar se cierra el archivo con close().

Si el usuario cancela la opción de guardar como, entonces se muestra un mensaje de advertencia indicando que puede perder los datos si no los guarda.

```
42 def guardarComo(event = None):
43     guardar_Como = filedialog.asksaveasfilename(initialdir=".", title="Guardar Como", filetypes=(("Archivo json", ".json"), ("all
44     files", "*.*")))
45     if guardar_Como != "":
46         saveComoArchivo = open(guardar_Como + ".json", "w") #+".json"
47         saveComoArchivo.write(textLeer.get("1.0", "end"))
48         saveComoArchivo.close()
49         showinfo(title="Guardado", message="Archivo guardado exitosamente")
50     else :
51         showwarning(title="Advertencia", message="¡Si no guarda el archivo se perderan los datos!")
```

### Método analizar

Inicia validando si existen datos en la variable almacenar, si la variable contiene datos, entonces accede al método analizando() de la clase AFD(), y se envía como parámetro a la variable con los datos, con el método analizandoSintacticamente() se realizan las validaciones correspondientes para hacer los cálculos y escribirlos en un archivo .dot, el cual se escribe con el método write(), luego con os.system convertimos a pdf el contenido del archivo .dot para crear la grafica, y con el método webbrowser.open\_new() se abre automáticamente el archivo pdf para que el usuario pueda visualizarlo correctamente.

```
53 def analizar(event = None):
54     #https://docs.python.org/es/3/reference/lexical_analysis.html
55     global almacenar
56     if almacenar != "":
57         try:
58             enviandoAnálisis.analizando(almacenar)
59             #enviandoAnálisis.imprimir_tokens()
60
61             operacionesGra = enviandoAnálisis.analizandoSintacticamente()
62             #ESCRIBIENDO .DOT
63             archivoGrafica = open(".\\Proyecto1\\RESULTADOS_201901103.dot", "w")
64             archivoGrafica.write(operacionesGra)
65             archivoGrafica.close()
66             os.system("dot -Tpdf .\\Proyecto1\\RESULTADOS_201901103.dot -o .\\Proyecto1\\RESULTADOS_201901103.pdf")
67             pathOperaciones = ".\\Proyecto1\\RESULTADOS_201901103.pdf"
68             webbrowser.open_new(pathOperaciones)
69         except Exception as e:
70             pass
71     else:
72         showwarning(title="Advertencia", message="Por favor cargue un archivo")
73
```

### Método errores

Se valida con if si la variable almacenar tiene datos, si contiene datos, entonces se accede al método erroresValidados() de la clase AFD(), el cual retornará una cadena con el archivo escrito con los datos de los errores encontrados, después de haber analizado el archivo.

Estos datos se muestran en el área de texto con el método insert(), y se desactiva dicha área para que el usuario no pueda editar en él. Luego con open indicamos en donde se guardará el archivo .json con los datos, y se escriben los datos con el método write() y

cerramos el archivo con close(). Además si la variable no contiene datos entonces se muestra un mensaje indicando que se cargue un archivo con anterioridad.

```
74  def errores(event = None):
75      global almacenar
76      if almacenar != "":
77          textErores.configure(state=tk.NORMAL)
78          escribiendoE = enviandoAnálisis.erroresValidados()
79          textErores.insert("1.0", escribiendoE)
80          textErores.configure(state="disabled")
81
82          archivoErrores = open(".\Proyecto1\ERRORES_201901103.json", "w")
83          archivoErrores.write(escribiendoE)
84          archivoErrores.close()
85
86      else:
87          showwarning(title="Advertencia", message="Por favor cargue un archivo")
```

### Método manual usuario

Este método tiene la finalidad de abrir el manual de usuario automáticamente, para que el usuario pueda visualizarlo, para ello se utiliza webbrowser.open\_new() y se envía como parámetro la ruta donde se encuentra el manual pdf.

```
91  def manualUsuario(event = None):
92      pathUsuario = ".\Proyecto1\Documentacion\Manual Usuario.pdf"
93      webbrowser.open_new(pathUsuario)
```

### Método manual técnico

Este método tiene la finalidad de abrir el manual técnico automáticamente, para que el usuario pueda visualizarlo, para ello se utiliza webbrowser.open\_new() y se envía como parámetro la ruta donde se encuentra el manual pdf.

```
95  def manualTecnico(event = None):
96      pathTecnico = ".\Proyecto1\Documentacion\Manual Tecnico.pdf"
97      webbrowser.open_new(pathTecnico)
```

### Método temas ayuda

Simplemente se muestra una ventana emergente informativa, indicando los datos del desarrollador de la aplicación.

```
99  def temasAyuda(event = None):
100     showinfo(title="Información del desarrollador", message="Josué Daniel Rojché García\nCarnet: 201901103\nLenguajes Formales y de Programación\nSección: A-")
```

## Clase operar

Se utiliza la librería math para realizar algunos cálculos, el método operando recibe como parámetros valor 1, valor 2 y el tipo de operación a realizar, dentro del método se validan con if que los datos no sean nulos, y si no lo son se verifica que el tipo de operación coincida con la que corresponde y si existe coincidencia se realiza la operación y se retorna el resultado.

```
Proyecto1 > operar.py > Operaciones
1 import math
2 class Operaciones:
3     def __init__(self):
4         pass
5
6     def operando(self, valor_1, valor_2, tipoOpera):
7         resultado = 0
8         if valor_1 != None:
9             valor_1 = float(valor_1)
10        if valor_2 != None:
11            valor_2 = float(valor_2)
12
13        if tipoOpera.lower() == 'suma':
14            resultado = valor_1 + valor_2
15            return resultado
16        elif tipoOpera.lower() == 'resta':
17            resultado = valor_1 - valor_2
18            return resultado
19        elif tipoOpera.lower() == 'multiplicacion':
20            resultado = valor_1 * valor_2
21            return resultado
22        elif tipoOpera.lower() == 'division':
23            try:
24                if valor_2 != 0:
25                    resultado = valor_1 / valor_2
26                else:
27                    return "Error division entre 0"
28            except ZeroDivisionError:
29                return "Error Error division entre 0"
30            else:
31                return resultado
32        elif tipoOpera.lower() == 'potencia':
33            resultado = pow(valor_1, valor_2)
34            return resultado
35        elif tipoOpera.lower() == 'raiz':
36
37            try:
38                if valor_1 >= 0:
39                    resultado = math.sqrt(valor_1)
40                else:
41                    return "No acepta negativos"
42            except Exception as e:
43                return e
44            else:
45                return resultado
46        elif tipoOpera.lower() == 'inverso':
47            try:
48                if valor_1 != 0:
49                    if valor_1 > 1:
50                        resultado = "1" + "/" + str(valor_1)
51                    elif valor_1 < -1:
52                        resultado = "-1" + "/" + str(valor_1 * -1)
53                    elif valor_1 < 0 and valor_1 > -1:
54                        resultado = 1/valor_1
55                    elif valor_1 > 0 and valor_1 < 1:
56                        resultado = 1/valor_1
57                else:
58                    return "Error no existe el inverso entre 0"
59            except ZeroDivisionError:
60                return "Error"
61            else:
62                return resultado
```

```
62        elif tipoOpera.lower() == 'seno':
63            #convirtiendo el angulo a radianes
64            convirtiendo = math.radians(valor_1)
65            resultado = math.sin(convirtiendo)
66            return resultado
67        elif tipoOpera.lower() == 'coseno':
68            #convirtiendo el angulo a radianes
69            convirtiendo1 = math.radians(valor_1)
70            resultado = math.cos(convirtiendo1)
71            return resultado
72        elif tipoOpera.lower() == 'tangente':
73            #convirtiendo el angulo a radianes
74            convirtiendo2 = math.radians(valor_1)
75            resultado = math.tan(convirtiendo2)
76            return resultado
77        elif tipoOpera.lower() == 'mod':
78            resultado = valor_1 % valor_2
79            return resultado
```



### Clase Token

En esta clase se ingresan fila, columna y lexema en el constructor, los cuales servirán para agregar en conjunto los datos a la tabla de tokens.

```
1 class Token:
2     def __init__(self, row, column, lexema):
3         self.row = row
4         self.column = column
5         self.lexema = lexema
```

### Clase AFD

En el método constructor se inicializa la lista de letras que podrá leer el automata, también la lista de números que podrá utilizarse, se inicializa las variables necesarias a utilizar, como fila, columna, los estados actual, anterior, estados finales, así como la lista para almacenar los tokens, y la lista para almacenar los errores, etc.

```
1 from token import Token
2 from operar import Operaciones
3
4 class AFD:
5     def __init__(self):
6         self.letras = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'ñ', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
7             'w', 'x', 'y', 'z', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '-']
8         self.numeros = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
9         self.fila = 0
10        self.columna = 0
11        self.estadoActual = ''
12        self.estadoAnterior = ''
13        self.estadoFinal = ['q10', 'q23']
14        self.tabla = []
15        self.tablaErrores = []
16        self.auxiliarTexto = ""
17        self.iterar = 1
18        self.escribiendoGrafica = ""
19        self.enlaceNodosSub = ""
```

### Método analizando

Este método recibe como parámetro el texto que se obtiene del archivo a analizar, el texto se almacena en una variable para manejar los datos localmente, por lo que se utiliza un bucle while para recorrer todos los datos del texto, luego se accede a cada carácter del texto y con un if se valida que se omitan los espacios, y saltos de línea que encuentre en el texto, para luego comenzar con las validaciones de if que indicaran en que estado se encuentra y que carácter puede almacenar en base a el diseño del Autómata Finito Determinista, el cual encuentra el diseño al final de este documento.

Básicamente lo que se realiza es repetitivo, solamente se valida que se encuentre en un estado en específico y luego cuando ingrese al carácter que coincida, envía el carácter como parámetro al método almacenarToken(), el cual sirve para enviar los datos a la lista de tokens. Luego se indica el estado actual como estado anterior, y el estado actual es

el siguiente estado del diseño del AFD, y con else validamos que si el carácter no es admitido por el autómata, entonces se trata de un error, por lo cual es necesario almacenar el carácter en el método almacenarError() y se utiliza una bandera que sirve para indicar que el error se ha encontrado y se quita hasta que encuentre nuevamente la bandera como false, en otro estado.

```

19 def analizando(self, texto1):
20     tok = ''
21     # Eliminando espacios y saltos de línea de la cadena
22     # texto = texto.replace("\n","")
23     # texto = texto.replace(" ","")
24     validandoError = False
25     texto = texto1
26     # recorriendo el texto
27     while len(texto) > 0:
28         caracter = texto[0]
29         if caracter == '\n':
30             self.fila += 1
31             self.columna = 0
32             texto = texto[1:]
33             continue
34         elif caracter == ' ':
35             self.columna += 1
36             texto = texto[1:]
37             continue
38
39     # validaciones de acuerdo al caracter que se está leyendo
40     # valida cuando inicia leyendo el texto
41     if self.estadoActual == '':
42         if caracter == '{':
43             self.almacenarToken(caracter)
44             self.estadoAnterior = 'q0'
45             self.estadoActual = 'q1'
46         else:
47             validandoError = True
48             self.almacenarError(caracter)
49     # valida cuando se encuentra en el estado q1

```

```

48         self.almacenarError(caracter)
49     # valida cuando se encuentra en el estado q1
50     if self.estadoActual == 'q1':
51         if caracter == '{':
52             self.almacenarToken(caracter)
53             self.estadoAnterior = 'q1'
54             self.estadoActual = 'q2'
55         elif caracter == '':
56             self.almacenarToken(caracter)
57             self.estadoAnterior = 'q1'
58             self.estadoActual = 'q3'
59         else:
60             validandoError = True
61             self.almacenarError(caracter)
62     # valida cuando se encuentra en q3 *****
63     elif self.estadoActual == 'q3':
64         if caracter.lower() in self.letras:
65             tok += caracter
66             self.estadoAnterior = 'q3'
67             self.estadoActual = 'q4'
68         else:
69             validandoError = True
70             self.almacenarError(caracter)
71     # valida cuando se encuentra en q4
72     elif self.estadoActual == 'q4':
73         if caracter.lower() in self.letras:
74             tok += caracter
75             self.estadoAnterior = 'q4'
76             self.estadoActual = 'q4'
77         elif caracter == '':
78             self.almacenarToken(tok)
79             tok = ''
80             self.almacenarToken(caracter)
81             self.estadoAnterior = 'q4'
82             self.estadoActual = 'q5'
83         else:
84             validandoError = True

```

```

82         self.estadoActual = 'q5'
83     else:
84         validandoError = True
85         self.almacenarError(caracter)
86     elif self.estadoActual == 'q5':
87         if caracter == ':':
88             self.almacenarToken(caracter)
89             self.estadoAnterior = 'q5'
90             self.estadoActual = 'q6'
91         else:
92             validandoError = True
93             self.almacenarError(caracter)
94     elif self.estadoActual == 'q6':
95         if caracter == '':
96             self.almacenarToken(caracter)
97             self.estadoAnterior = 'q6'
98             self.estadoActual = 'q7'
99         else:
100             validandoError = True
101             self.almacenarError(caracter)
102     elif self.estadoActual == 'q7':
103         if caracter.lower() in self.letras:
104             tok += caracter
105             self.estadoAnterior = 'q7'
106             self.estadoActual = 'q8'
107         else:
108             validandoError = True
109             self.almacenarError(caracter)
110     # valida cuando se encuentra en q8
111     elif self.estadoActual == 'q8':
112         if caracter.lower() in self.letras:
113             tok += caracter
114             self.estadoAnterior = 'q8'
115             self.estadoActual = 'q8'

```

```

115         self.estadoActual = 'q8'
116     elif caracter == '':
117         self.almacenarToken(tok)
118         tok = ''
119         self.almacenarToken(caracter)
120         self.estadoAnterior = 'q8'
121         self.estadoActual = 'q9'
122     else:
123         validandoError = True
124         self.almacenarError(caracter)
125     # valida cuando se encuentra en q9
126     elif self.estadoActual == 'q9':
127         if caracter == ',':
128             self.almacenarToken(caracter)
129             self.estadoAnterior = 'q9'
130             self.estadoActual = 'q1'
131             validandoError = False
132         elif caracter == '}':
133             # estado de aceptación
134             self.almacenarToken(caracter)
135             self.estadoAnterior = 'q9'
136             self.estadoActual = 'q10'
137             validandoError = False
138         else:
139             validandoError = True
140             self.almacenarError(caracter)
141     # Aquí termina una parte *****
142     # Segunda parte
143     # validar estado q2
144     elif self.estadoActual == 'q2':
145         if caracter == '':
146             self.almacenarToken(caracter)
147             self.estadoAnterior = 'q2'
148             self.estadoActual = 'q11'
149         else:
150             validandoError = True
151             self.almacenarError(caracter)

```

```

152     # validacion estado q11
153     elif self.estadoActual == 'q11':
154         if caracter.lower() in self.letras:
155             tok += caracter
156             self.estadoAnterior = 'q11'
157             self.estadoActual = 'q12'
158         else:
159             validandoError = True
160             self.almacenarError(caracter)
161     # validando estado q12
162     elif self.estadoActual == 'q12':
163         if caracter.lower() in self.letras:
164             tok += caracter
165             self.estadoAnterior = 'q12'
166             self.estadoActual = 'q12'
167         elif caracter == '':
168             self.almacenarToken(tok)
169             tok = ''
170             self.almacenarToken(caracter)
171             self.estadoAnterior = 'q12'
172             self.estadoActual = 'q13'
173         else:
174             validandoError = True
175             self.almacenarError(caracter)
176     # validando estado q13
177     elif self.estadoActual == 'q13':
178         if caracter == ':':
179             self.almacenarToken(caracter)
180             self.estadoAnterior = 'q13'
181             self.estadoActual = 'q14'
182         else:
183             validandoError = True
184             self.almacenarError(caracter)
185     # validando estado q14
186     elif self.estadoActual == 'q14':
187         if caracter in self.numeros:
188             tok += caracter

```

```

187         if caracter in self.numeros:
188             tok += caracter
189             self.estadoAnterior = 'q14'
190             self.estadoActual = 'q18'
191         elif caracter == '':
192             self.almacenarToken(caracter)
193             self.estadoAnterior = 'q14'
194             self.estadoActual = 'q15'
195         elif caracter == '[':
196             self.almacenarToken(caracter)
197             self.estadoAnterior = 'q14'
198             self.estadoActual = 'q2'
199         else:
200             validandoError = True
201             self.almacenarError(caracter)
202     # validando estado q15
203     elif self.estadoActual == 'q15':
204         if caracter.lower() in self.letras:
205             tok += caracter
206             self.estadoAnterior = 'q15'
207             self.estadoActual = 'q16'
208         else:
209             validandoError = True
210             self.almacenarError(caracter)
211     # validando estado q16
212     elif self.estadoActual == 'q16':
213         if caracter.lower() in self.letras:
214             tok += caracter
215             self.estadoAnterior = 'q16'
216             self.estadoActual = 'q16'
217         elif caracter == '':
218             self.almacenarToken(tok)
219             tok = ''
220             self.almacenarToken(caracter)
221             self.estadoAnterior = 'q16'
222             self.estadoActual = 'q17'

```

El código continúa hasta aumentar la columna, eliminar el carácter analizado de la cadena de texto y retornar el estado actual si corresponde al estado de aceptación.

```
319 # validando estado q22
320 elif self.estadoActual == 'q22':
321     if caracter == ',':
322         self.almacenarToken(caracter)
323         self.estadoAnterior = 'q22'
324         self.estadoActual = 'q1'
325         validandoError = False
326         # estado de aceptación
327     elif caracter == '}':
328         self.almacenarToken(caracter)
329         self.estadoAnterior = 'q22'
330         self.estadoActual = 'q23'
331     else:
332         validandoError = True
333         self.almacenarError(caracter)
334 # validando estado q24
335 elif self.estadoActual == 'q24':
336     if caracter == ']':
337         self.almacenarToken(caracter)
338         self.estadoAnterior = 'q24'
339         self.estadoActual = 'q24'
340     elif caracter == ',':
341         self.almacenarToken(caracter)
342         self.estadoAnterior = 'q24'
343         self.estadoActual = 'q2'
344     elif caracter == '}':
345         self.almacenarToken(caracter)
346         self.estadoAnterior = 'q24'
347         self.estadoActual = 'q22'
348     else:
349         validandoError = True
350         self.almacenarError(caracter)
351
352 self.columna += 1
353 texto = texto[1:]
354 return self.estadoActual in self.estadoFinal
```

### Método almacenarToken

Recibe como parámetro el lexema y se crea una instancia para almacenar la fila y columna que corresponde al lexema encontrado en el automata, luego este se almacena en la tabla de tokens.

```
356 def almacenarToken(self, lexema):
357     newToken = Token(self.fila, self.columna, lexema)
358     self.tabla.append(newToken)
```

### Método almacenarError

Recibe como parámetro el lexema y se crea una instancia para almacenar la fila y columna que corresponde al lexema encontrado en el autómata, luego este se almacena en la tabla de errores.

```
748  def almacenarError(self, lexemaError):
749      newToken1 = Token(self.fila, self.columna, lexemaError)
750      self.tablaErrores.append(newToken1)
```

### Método erroresValidados

Este método sirve para acceder a la tabla de errores y escribir en una variable la estructura para el archivo de errores .json que requiere generar el usuario en la opción de errores, por lo que simplemente se accede a cada dato y se agrega al espacio que corresponde, luego se elimina la ultima coma y se retorna la cadena.

```
368  def erroresValidados(self):
369      escribiendoEstructura = "{\n"
370      iterador = 1
371      for token in self.tablaErrores:
372          escribiendoEstructura = escribiendoEstructura + "\t{\n"
373          escribiendoEstructura = escribiendoEstructura + \
374              '\t\t"No.": ' + str(iterador) + "\n"
375          escribiendoEstructura = escribiendoEstructura + \
376              '\t\t"Descripcion-Token":{\n'
377          escribiendoEstructura = escribiendoEstructura + \
378              '\t\t\t"Lexema": ' + str(token.lexema) + '\n'
379          escribiendoEstructura = escribiendoEstructura + '\t\t\t"Tipo": Error\n'
380          escribiendoEstructura = escribiendoEstructura + \
381              '\t\t\t"Columna": ' + str(token.columna) + '\n'
382          escribiendoEstructura = escribiendoEstructura + \
383              '\t\t\t"Fila": ' + str(token.row) + '\n'
384          escribiendoEstructura = escribiendoEstructura + "\t\t}\n"
385          escribiendoEstructura = escribiendoEstructura + "\t},\n"
386          iterador += 1
387          # print(token.row, token.columna, token.lexema)
388          total = escribiendoEstructura[:-2] + "\n}"
389          # escribiendoEstructura = total + "\n}"
390          return total
```

### Método analizandoSintacticamente

Este método sirve para obtener los datos de la tabla de tokens y validar que se realicen las operaciones conforme la jerarquía con la que fue ingresado el archivo, respetando la estructura del mismo, por lo que se utiliza un bucle while para recorrer toda la tabla, sin embargo también se requiere de escribir el archivo .dot con los datos obtenidos de los cálculos, para así mostrarlos en forma gráfica. En las validaciones if podrá observar que se va validando las opciones para las asignaciones de los valores que corresponden a sus determinados campos, sin embargo cuando se llega hasta el if donde se obtiene un

valor anidado se comienza a ingresar a un método que puede hacerse recursivo si es necesario validar otras operaciones que sean anidadas, de tal manera que se realicen todas las operaciones de manera ordenada y correctamente. Esto es algo que se repite en todo el método, por lo cual solo se agregan las otras imágenes.

```

394 def analizandoSintacticamente(self):
395     self.iniciandoEsGrafica = '''digraph G {
396         rankdir=LR
397         size= 8.5
398         ranksep=1
399         bgcolor = lightgoldenrod2yellow
400         margin = 0.1
401         edge[arrowhead = vee];'''
402     self.escribiendoGrafica = ""
403     calculando = Operaciones()
404     # recorrer la tabla y ir validando con ifs anidados para ver los datos que están como frases o como numeros
405     self.iterar = 1
406     # valida si viene un corchete
407     while self.iterar < len(self.tabla):
408         if self.tabla[self.iterar].lexema == "{":
409             print(self.tabla[self.iterar].lexema)
410             self.iterar += 1
411             # valida si el siguiente es una comilla
412             if self.tabla[self.iterar].lexema == "'":
413                 self.iterar += 1
414                 if self.tabla[self.iterar].lexema.lower() == 'operacion':
415                     asignacion_Operacion = ""
416                     self.iterar += 1
417                     # valida si el que finaliza es una comilla entonces se creó la variable operacion
418                     if self.tabla[self.iterar].lexema == "'":
419                         self.iterar += 1
420                         if self.tabla[self.iterar].lexema == ":":
421                             # aqui se valida si lo ve sigue son comillas para la asignacion de la variable
422                             self.iterar += 1
423                             if self.tabla[self.iterar].lexema == "'":
424                                 self.iterar += 1
425                                 asignacion_Operacion = self.tabla[self.iterar].lexema
426                                 print(asignacion_Operacion)
427                                 self.iterar += 1
428                                 if self.tabla[self.iterar].lexema == "'":
429                                     self.iterar += 1

```

```

430             if self.tabla[self.iterar].lexema == ',':
431                 self.iterar += 1
432             if self.tabla[self.iterar].lexema == "'":
433                 self.iterar += 1
434                 if self.tabla[self.iterar].lexema.lower() == 'valor1':
435                     valor1 = ""
436                     self.iterar += 1
437                     if self.tabla[self.iterar].lexema == "'":
438                         self.iterar += 1
439                         if self.tabla[self.iterar].lexema == ":":
440                             self.iterar += 1
441                             if self.tabla[self.iterar].lexema == '[':
442                                 resultado_valor1 = self.operacionAnidada()
443                                 enlace1 = self.enlaceNodosSub
444                                 print(resultado_valor1)
445                                 if self.tabla[self.iterar].lexema == ']':
446                                     self.iterar += 1
447                                     if self.tabla[self.iterar].lexema == ',':
448                                         self.iterar += 1
449                                         if self.tabla[self.iterar].lexema == "'":
450                                             self.iterar += 1
451                                             if self.tabla[self.iterar].lexema.lower() == 'valor2':
452                                                 valor2 = ""
453                                                 self.iterar += 1
454                                                 if self.tabla[self.iterar].lexema == "'":
455                                                     self.iterar += 1
456                                                     if self.tabla[self.iterar].lexema == ':':
457                                                         self.iterar += 1
458                                                         if self.tabla[self.iterar].lexema == '[':
459                                                             resultado_valor2 = self.operacionAnidada()
460                                                             print(resultado_valor2)
461                                                             resultadoCalculos = calculando.operando(resultado_valor1, resultado_valor2,
462                                                                 asignacion_Operacion)
463                                                             print("Resultado: " + str(resultadoCalculos))
464                                                             enlace2 = self.enlaceNodosSub
465                                                             self.escribiendoGrafica = self.escribiendoGrafica + '''
a0 [label = ''' + asignacion_Operacion + '\n''' + str(resultadoCalculos) + '''];

```

```

466 ~ a0 ->''' + enlace1 +''';
467 ~ a0 ->''' + enlace2 +''';'''
468 ~
469 ~ if self.tabla[self.iterar].lexema == ']':
470 ~     self.iterar += 1
471 ~     if self.tabla[self.iterar].lexema == ']':
472 ~         self.iterar += 1
473 ~         elif self.tabla[self.iterar].lexema == '':
474 ~             print(self.tabla[self.iterar].lexema)
475 ~             self.iterar += 1
476 ~     else:
477 ~         valor2 = self.tabla[self.iterar].lexema
478 ~         print(valor2)
479 ~         self.iterar += 1
480 ~         resultadoCalculos = calculando.operando(resultado_valor1, valor2,
481 ~             asignacion_Operacion)
482 ~
483 ~         print("Resultado: " + str(resultadoCalculos))
484 ~
485 ~         self.escribiendoGrafica = self.escribiendoGrafica + '''
486 ~
487 ~         b0 [label = ''' + asignacion_Operacion + '''\n''' + str(resultadoCalculos) +'''];
488 ~         b2 [label = ''' + str(valor2) +'''];
489 ~         b0 ->''' + enlace1 +''';
490 ~         b0 -> b2;'''
491 ~
492 ~         if self.tabla[self.iterar].lexema == '':
493 ~             print(self.tabla[self.iterar].lexema)
494 ~             self.iterar += 1
495 ~         else:
496 ~             valor1 = self.tabla[self.iterar].lexema
497 ~             print(valor1)
498 ~             self.iterar += 1
499 ~             if self.tabla[self.iterar].lexema == ',':
500 ~                 self.iterar += 1
501 ~                 if self.tabla[self.iterar].lexema == '':
502 ~                     self.iterar += 1
503 ~                     if self.tabla[self.iterar].lexema.lower() == 'valor2':
504 ~                         valor2 = ""

```

```

502 ~ self.iterar += 1
503 ~ if self.tabla[self.iterar].lexema == '':
504 ~     self.iterar += 1
505 ~     if self.tabla[self.iterar].lexema == ':':
506 ~         self.iterar += 1
507 ~         if self.tabla[self.iterar].lexema == '[':
508 ~             resultado_valor2 = self.operacionAnidada()
509 ~             print(resultado_valor2)
510 ~             resultadoCalculos = calculando.operando(valor1, resultado_valor2,
511 ~                 asignacion_Operacion)
512 ~             print("Resultado: " + str(resultadoCalculos))
513 ~             enlace2 = self.enlaceNodosSub
514 ~             self.escribiendoGrafica = self.escribiendoGrafica + '''
515 ~
516 ~             c0 [label = ''' + asignacion_Operacion + '''\n''' + str(resultadoCalculos) +'''];
517 ~             c1 [label = ''' + str(valor1) +'''];
518 ~             c0 -> c1;
519 ~             c0 ->''' + enlace2 +''';'''
520 ~
521 ~             if self.tabla[self.iterar].lexema == ']':
522 ~                 self.iterar += 1
523 ~                 if self.tabla[self.iterar].lexema == '':
524 ~                     print(self.tabla[self.iterar].lexema)
525 ~                     self.iterar += 1
526 ~                 else:
527 ~                     valor2 = self.tabla[self.iterar].lexema
528 ~                     print(valor2)
529 ~                     resultadoCalculos = calculando.operando(valor1, valor2, asignacion_Operacion)
530 ~                     print("Resultado: " + str(resultadoCalculos))
531 ~
532 ~                     self.escribiendoGrafica = self.escribiendoGrafica + '''
533 ~
534 ~                     d0 [label = ''' + asignacion_Operacion + '''\n''' + str(resultadoCalculos) +'''];
535 ~                     d1 [label = ''' + str(valor1) +'''];
536 ~                     d2 [label = ''' + str(valor2) +'''];
537 ~                     d0 -> d1;
538 ~                     d0 -> d2;'''
539 ~
540 ~                     self.iterar += 1
541 ~                     if self.tabla[self.iterar].lexema == '':
542 ~                         print(self.tabla[self.iterar].lexema)

```





llevar el orden de la jerarquía de las operaciones a realizar conforme el archivo leído. Y al encontrar el que corresponde automáticamente retorna el resultado al método anterior.

```

611 def operacionAnidada(self):
612     calculando = Operaciones()
613     self.iterar +=1
614     #valida si el siguiente es una comilla
615     if self.tabla[self.iterar].lexema == '':
616         self.iterar +=1
617         if self.tabla[self.iterar].lexema.lower() == 'operacion':
618             asignacion_Operacion = ""
619             self.iterar +=1
620             #valida si el que finaliza es una comilla entonces se creó la variable operacion
621             if self.tabla[self.iterar].lexema == '':
622                 self.iterar +=1
623                 if self.tabla[self.iterar].lexema == ':':
624                     #aqui se valida si lo ve sigue son comillas para la asignacion de la variable
625                     self.iterar +=1
626                     if self.tabla[self.iterar].lexema == '':
627                         self.iterar +=1
628                         asignacion_Operacion = self.tabla[self.iterar].lexema
629                         print(asignacion_Operacion)
630                         self.iterar +=1
631                         if self.tabla[self.iterar].lexema == '':
632                             self.iterar +=1
633                             if self.tabla[self.iterar].lexema == ',':
634                                 self.iterar +=1
635                                 if self.tabla[self.iterar].lexema == '':
636                                     self.iterar +=1
637                                     if self.tabla[self.iterar].lexema.lower() == 'valor1':
638                                         valor1 = ""
639                                         self.iterar += 1
640                                         if self.tabla[self.iterar].lexema == '':
641                                             self.iterar +=1
642                                             if self.tabla[self.iterar].lexema == ':':
643                                                 self.iterar +=1
644                                                 if self.tabla[self.iterar].lexema == '[':
645                                                     resultado_valor1 = self.operacionAnidada()
646                                                     enlace1 = self.enlaceNodosSub
647                                                     print(resultado_valor1)
648                                                     if self.tabla[self.iterar].lexema == ']':
649                                                         self.iterar += 1
650                                                         if self.tabla[self.iterar].lexema == ',':
651                                                             self.iterar += 1
652                                                             if self.tabla[self.iterar].lexema == '':
653                                                                 self.iterar += 1
654                                                                 if self.tabla[self.iterar].lexema.lower() == 'valor2':
655                                                                     valor2 = ""
656                                                                     self.iterar += 1
657                                                                     if self.tabla[self.iterar].lexema == '':
658                                                                         self.iterar += 1
659                                                                         if self.tabla[self.iterar].lexema == ':':
660                                                                             self.iterar += 1
661                                                                             if self.tabla[self.iterar].lexema == '[':
662                                                                                 resultado_valor2 = self.operacionAnidada()
663                                                                                 print(resultado_valor2)
664                                                                                 resultadoCalculos = calculando.operando(resultado_valor1, resultado_valor2,
665                                                                                                     asignacion_Operacion)
666                                                                                 print("Resultado: " + str(resultadoCalculos))
667                                                                                 enlace2 = self.enlaceNodosSub
668                                                                                 self.escribiendoGrafica = self.escribiendoGrafica + ' '
669                                     e0 [label = ""'+asignacion_Operacion+''\n'+str(resultadoCalculos)+'"];
670                                     e0 ->''+ enlace1 + ' ';
671                                     e0 ->''+ enlace2 + ' ';
672                                     self.enlaceNodosSub = "e0"
673                                     return resultadoCalculos
674                                 else:
675                                     valor2 = self.tabla[self.iterar].lexema
676                                     print(valor2)
677                                     self.iterar += 1
678                                     resultadoCalculos = calculando.operando(resultado_valor1, valor2,
679                                                                                     asignacion_Operacion)
680                                     print("Resultado: " + str(resultadoCalculos))
681                                     self.escribiendoGrafica = self.escribiendoGrafica + ' '
682                                     f0 [label = ""'+asignacion_Operacion+''\n'+str(resultadoCalculos)+'"];

```

```

683 ~ f2 [label = ""'+str(valor2)+'"];
684 f0 ->''+ enlace1 +'';
685 f0 -> f2;'''
686 self.enlaceNodosSub = "f0"
687 return resultadoCalculos
688 else:
689     resultadoCalculos = calculando.operando(resultado_valor1, None, asignacion_Operacion)
690     print("Resultado: " + str(resultadoCalculos))
691     self.escribiendoGrafica = self.escribiendoGrafica + '''
692 g0 [label = ""'+asignacion_Operacion+''\n'''+str(resultadoCalculos)+'"];
693 g0 ->''+ enlace1 +'';'''
694 self.enlaceNodosSub = "g0"
695 return resultadoCalculos
696 else:
697     valor1 = self.tabla[self.iterar].lexema
698     print(valor1)
699     self.iterar +=1
700 if self.tabla[self.iterar].lexema == ',':
701     self.iterar +=1
702 if self.tabla[self.iterar].lexema == '':
703     self.iterar +=1
704 if self.tabla[self.iterar].lexema.lower() == 'valor2':
705     valor2 = ""
706     self.iterar += 1
707 if self.tabla[self.iterar].lexema == '':
708     self.iterar +=1
709 if self.tabla[self.iterar].lexema == '[':
710     self.iterar +=1
711 if self.tabla[self.iterar].lexema == '[':
712     resultado_valor2 = self.operacionAnidada()
713     print(resultado_valor2)
714     resultadoCalculos = calculando.operando(valor1, resultado_valor2, asignacion_Operacion)
715     print("Resultado: " + str(resultadoCalculos))
716     enlace2 = self.enlaceNodosSub
717     self.escribiendoGrafica = self.escribiendoGrafica + '''
718 h0 [label = ""'+asignacion_Operacion+''\n'''+str(resultadoCalculos)+'"];
719 h1 [label = ""'+str(valor1)+'"];

```

```

720 h0 -> h1;
721 h0 ->''+ enlace2 +'';'''
722 self.enlaceNodosSub = "h0"
723 return resultadoCalculos
724 else:
725     valor2 = self.tabla[self.iterar].lexema
726     print(valor2)
727     resultadoCalculos = calculando.operando(valor1, valor2, asignacion_Operacion)
728     print("Resultado: " + str(resultadoCalculos))
729     self.escribiendoGrafica = self.escribiendoGrafica + '''
730 i0 [label = ""'+asignacion_Operacion+''\n'''+str(resultadoCalculos)+'"];
731 i1 [label = ""'+str(valor1)+'"];
732 i2 [label = ""'+str(valor2)+'"];
733 i0 -> i1;
734 i0 -> i2;'''
735 self.enlaceNodosSub = "i0"
736 self.iterar +=1
737 return resultadoCalculos
738 else:
739     resultadoCalculos = calculando.operando(valor1, None, asignacion_Operacion)
740     print("Resultado: " + str(resultadoCalculos))
741     self.escribiendoGrafica = self.escribiendoGrafica + '''
742 j0 [label = ""'+asignacion_Operacion+''\n'''+str(resultadoCalculos)+'"];
743 j1 [label = ""'+str(valor1)+'"];
744 j0 -> j1;'''
745 self.enlaceNodosSub = "j0"
746 return resultadoCalculos

```

## Autómata Finito Determinista

En la imagen siguiente se ilustra el autómata finito determinista que fue diseñado para darle solución a la lectura correcta del archivo a analizar en el software.

