MANUAL TECNICO:

Flujo del sistema

- De primero cargamos los archivos
- Tenemos la decisión de tipo de archivo (por medio de botones)
- Análisis del sistema
- Recopilación de errores
- Reubicación del archivo
- Verificación de resultados

Codigo de vistas del programa:

Este código es para toda las vistas del programa teniendo métodos donde se hace la importación de las distintas clases del programa para poder implementar cada analizador del sistema de una forma visual con una interface grafica.

```
# arreglar problema de importar y que no lo ejecute de una vez, aunque funciono p
oniendo el import en el metodo
# import AnalisisLexicoHtml
# import AnalisadorLexicoCss

# from tkinter import *    # Carga módulo tk (widgets estándar)
# from tkinter import ttk  # Carga ttk (para widgets nuevos 8.5+)
import os, sys
from tkinter import Tk, Menu, messagebox, filedialog, ttk, Label, scrolledtext, I
NSERT, END, Button, Scrollbar, RIGHT, Y, Frame, Canvas, HORIZONTAL, VERTICAL, sim
pledialog

root = Tk()
root.title("PROYECTO 1 ANALIZADORE LEXICOS")
root.configure(background = "yellow")

'''FUNCIONES DEL MENU'''

archivo = ""

def nuevo():
    global archivo
    editor.delete(1.0, END)#ELIMINAR EL CONTENIDO
    editor2.delete(1.0, END)
    archivo = ""
```

```
def abrir():
    global archivo
    archivo = filedialog.askopenfilename(title = "Abrir Archivo", initialdir = "C
:/")

    entrada = open(archivo,"r",encoding="utf-8")
    content = entrada.read()

    editor.delete(1.0, END)
    editor.insert(INSERT, content)
    entrada.close()

def salir():
    value = messagebox.askokcancel("Salir", "Está seguro que desea salir?")
    if value :
        root.destroy()

def guardarArchivo():
    global archivo
    if archivo == "":
        guardarComo()
    else:
        guardarc = open(archivo, "w")
        guardarc.write(editor.get(1.0, END))
        guardarc.close()

def guardarComo():
    global archivo
    guardar = filedialog.asksaveasfilename(title = "Guardar Archivo", initialdir
= "C:/")
    fguardar = open(guardar, "w+")
    fguardar.write(editor.get(1.0, END))
    fguardar.close()
    archivo = guardar

def analisarhtml ():
    from AnalisisLexicoHtml import AnalisisLexicoHtml
    holaH = AnalisisLexicoHtml()
    entrada2 = open (archivo,"r",encoding="utf-8")
    contenidoH = entrada2.read()
    tokensH = holaH.inic(contenidoH)
    entrada2.close()
    for token in tokensH:
        print(token)
    #generar el archivo htm de los errores lexicos del archivo de entrada
```

```
    salidaH=''
    salidaH += '<html>\n'
    salidaH += '<head\n>'
    salidaH +='<title>archivo de JS</title>\n'
    salidaH +='</head> \n <body style="background-
color:black"> \n <h1 style="text-
align: center;color:white"> errores lexicos HTML <h1>\n'
    salidaH +='<div class="container">\n <h2 style="color:white">ERRORES</h2>\n <
div style="background-color:white">\n'
    salidaH +='<table> \n <thead> \n <tr> \n <th>linea</th>\n <th>columna</th>\n
<th> token </th>\n </tr>\n'

    for error in AnalisisLexicoHtml.Errores:
        salidaH +='<tr>\n'
        salidaH += '<th>' + str(error[0]) +'</th>\n'
        salidaH += '<th>' + str(error[1]) +'</th>\n'
        salidaH += '<th>' + error[2] +'</th>\n'
        salidaH +='</tr>\n'
    salidaH +='</thead>\n </table>\n </div>\n </div>\n <footer style="background-
color: black; color:white">\n <p>FIN DE LA TABLA DE ERRORES</p>\n'
    salidaH +='</footer>\n </body> \n </html>'
    print (salidaH)
    HTML = open ('analisishtml.html','w')
    HTML.write (salidaH)
    HTML.close()
    #guargar el html de errores en el path que nos da el archivo de entrada
    # pathh= AnalisisLexicoHtml.Comentarios[0]
    # os.mkdir(path[pathh[3].replace('PATHW:',' '), mode])
    # os.mkdir( pathh[3].replace('PATHW:',' '), 0o777)
    # guardarc = open(pathh[3].replace('PATHW:',' '), "w",encoding="utf-8")
    # guardarc.write(salidaH)
    # guardarc.close()

def analisarCss ():
    from AnalisadorLexicoCss import AnalisadorLexicoCss
    hola = AnalisadorLexicoCss()
    entrada2 = open(archivo,"r",encoding="utf-8")
    content = entrada2.read()
    tokens = hola.inic(content)
    for token in tokens:
        print(token)
    print ('BITACORA')
    for bita in AnalisadorLexicoCss.Bitacora:
        editor2.insert(INSERT,[bita[1],bita[2],bita[0],bita[3]+'\n'])
        print (bita)
```

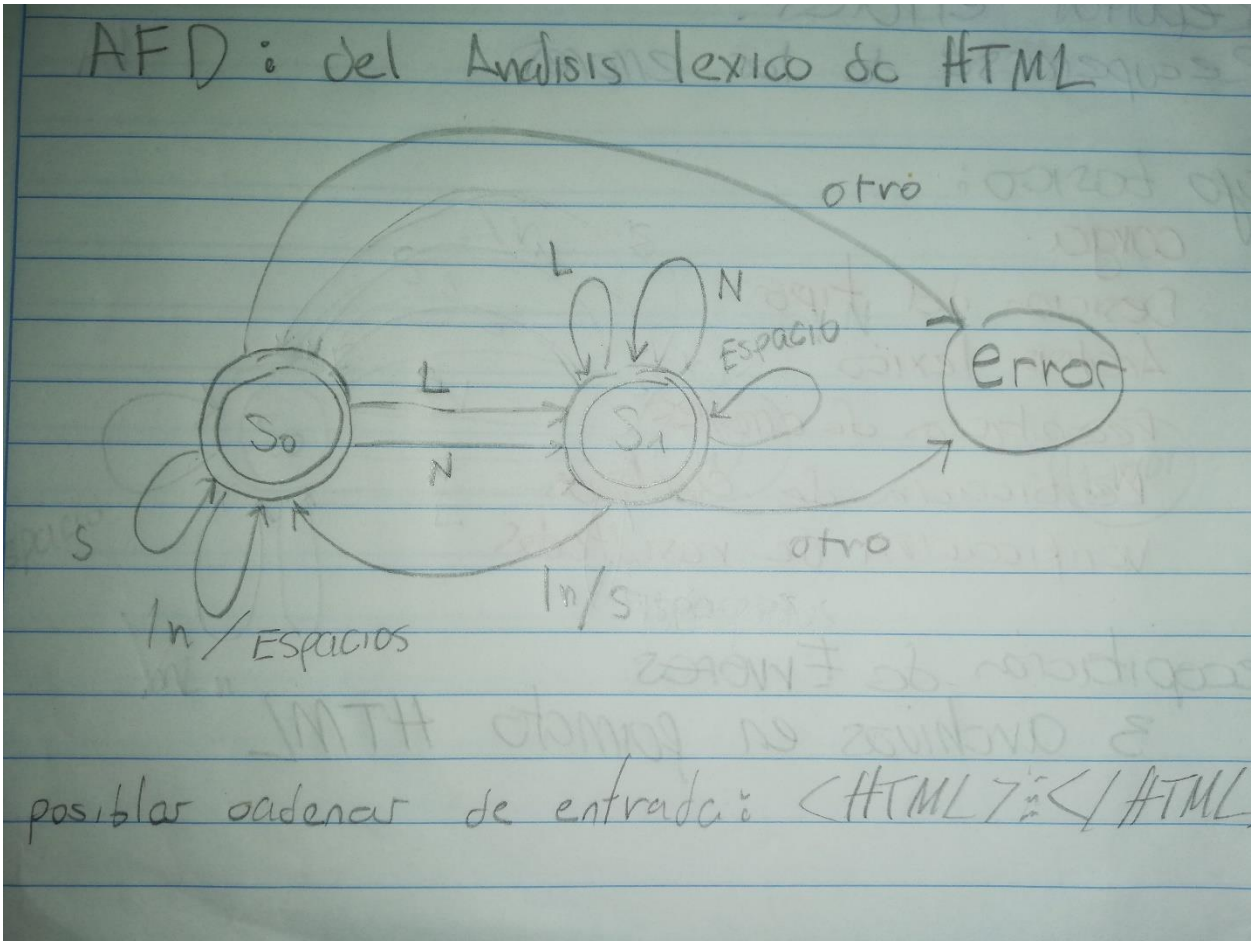
```
    entrada2.close()
    salidaH=''
    salidaH += '<html>\n'
    salidaH += '<head\n>'
    salidaH +='<title>archivo de JS</title>\n'
    salidaH +='</head> \n <body style="background-
color:black"> \n <h1 style="text-
align: center;color:white"> errores lexicos CSS <h1>\n'
    salidaH +='<div class="container">\n <h2 style="color:white">ERRORES</h2>\n <
div style="background-color:white">\n'
    salidaH +='<table> \n <thead> \n <tr> \n <th>linea</th>\n <th>columna</th>\n
<th> token </th>\n </tr>\n'
    for error in AnalisadorLexicoCss.Errores:
        salidaH +='<tr>\n'
        salidaH += '<th>' + str(error[0]) +'</th>\n'
        salidaH += '<th>' + str(error[1]) +'</th>\n'
        salidaH += '<th>' + error[2] +'</th>\n'
        salidaH +='</tr>\n'
    salidaH +='</thead>\n </table>\n </div>\n </div>\n <footer style="background-
color: black; color:white">\n <p>FIN DE LA TABLA DE ERRORES</p>\n'
    salidaH +='</footer>\n </body> \n </html>'
    print (salidaH)
    HTML = open ('analisisCSS.html','w')
    HTML.write (salidaH)
    HTML.close()

def analisisJs():
    from AnalisadorLexicoJs import AnalisadorLexicoJs
    holaJs = AnalisadorLexicoJs()
    entrada3 = open(archivo,"r",encoding="utf-8")
    content = entrada3.read()
    tokens = holaJs.inic(content)
    for token in tokens:
        print(token)
    for error in AnalisadorLexicoJs.Errores:
        editor2.insert(INSERT,[error[0],error[1],error[2]+'\n'])
    for coment in AnalisadorLexicoJs.Comentarios:
        editor2.insert(INSERT,[coment[0],coment[1],coment[2],coment[3]+'\n'])
    entrada3.close()
    from pruebaGra import grafica
    salidaH=''
    salidaH += '<html>\n'
    salidaH += '<head\n>'
    salidaH +='<title>archivo de JS</title>\n'
```

```
    salidaH +='</head> \n <body style="background-
color:black"> \n <h1 style="text-
align: center;color:white"> errores lexicos JS <h1>\n'
    salidaH +='<div class="container">\n <h2 style="color:white">ERRORES</h2>\n <
div style="background-color:white">\n'
    salidaH +='<table> \n <thead> \n <tr> \n <th>linea</th>\n <th>columna</th>\n
<th> token </th>\n </tr>\n'
    for error in AnalisadorLexicoJs.Errores:
        salidaH +='<tr>\n'
        salidaH += '<th>' + str(error[0]) +'</th>\n'
        salidaH += '<th>' + str(error[1]) +'</th>\n'
        salidaH += '<th>' + error[2] +'</th>\n'
        salidaH +='</tr>\n'
    salidaH +='</thead>\n </table>\n </div>\n </div>\n <footer style="background-
color: black; color:white">\n <p>FIN DE LA TABLA DE ERRORES</p>\n'
    salidaH +='</footer>\n </body> \n </html>'
    print (salidaH)
    HTML = open ('analisisJS.html','w')
    HTML.write (salidaH)
    HTML.close()

def analisisCalcu():
    from AnalisisCalcu import AnalisadorLexico
    from AnalisisCalcu import sintac
    holaCalcu = AnalisadorLexico()
    entrada4 = open(archivo,"r",encoding="utf-8")
    content = entrada4.read()
    tokens = holaCalcu.inic(content)
    motor = sintac()
    correccion = motor.parse(tokens)
    if correccion:
        editor2.insert(INSERT,'CORRECTO \n')
        print("Analisis Sintactico Correcto.")
    else:
        editor2.insert(INSERT,'INCORRECTO \n')
        print("Analisis Sintactico Incorrecto.")


# MENU Y BARRA DE OPCIONES
barraMenu = Menu(root)
root.config(menu = barraMenu, width = 1000, height = 600)

archivoMenu = Menu(barraMenu, tearoff=0)
archivoMenu.add_command(label = "Nuevo", command = nuevo)
archivoMenu.add_command(label = "Abrir", command = abrir)
```

```
archivoMenu.add_command(label = "Guardar", command = guardarArchivo)
archivoMenu.add_command(label = "Guardar Como...", command = guardarComo)
archivoMenu.add_separator()
archivoMenu.add_command(label = "Salir", command = salir)

archivoMenu2 = Menu(barraMenu, tearoff=0)
archivoMenu2.add_command(label = "Analisis HTML", command = analisarhtml)
archivoMenu2.add_command(label = "Analisis CSS", command = analisarCss)
archivoMenu2.add_command(label = "Analisis JAVASCRIPT", command = analisisJs)
archivoMenu2.add_command(label = "Analisis Calculadora", command = analisisCalcu)

barraMenu.add_cascade(label = "Archivo", menu = archivoMenu)
barraMenu.add_cascade(label = "Analisadores", menu = archivoMenu2)
barraMenu.add_command(label = "Salir",  command = salir)

frame = Frame(root, bg="green")
canvas = Canvas(frame, bg="green")
scrollbar = Scrollbar(frame, orient=VERTICAL, command=canvas.yview)
scroll = Frame(canvas, bg="black")

scroll.bind("<Configure>",lambda e: canvas.configure(scrollregion=canvas.bbox("al
l")))

canvas.create_window((0, 0), window=scroll, anchor="nw")

canvas.configure(yscrollcommand=scrollbar.set, width = 1300, height = 600)

ttk.Label(scroll, text = "ANALISIS LEXICO", font = ("Arial", 17), background='bla
ck', foreground = "white").grid(column = 1, row = 0)

editor = scrolledtext.ScrolledText(scroll, undo = True, width = 50, height = 20,
font = ("Arial", 15), background = 'white',  foreground = "black")

editor.grid(column = 1, row = 1, pady = 25, padx = 10)


editor2 = scrolledtext.ScrolledText(scroll, undo = True, width = 50, height = 20,
 font = ("Arial", 15), background = 'yellow',  foreground = "black")

editor2.grid(column = 5, row = 1, pady = 25, padx = 10)

frame.grid(sticky='news')
canvas.grid(row=0,column=1)
scrollbar.grid(row=0, column=2, sticky='ns')
```

```
editor.focus()
root.mainloop()
```

FOTOGRAFICA DEL AUTOMATA DISEñADO PARA LA IMPLEMENTACION DE CADA ANALISADOR LEXICO:



AUTOMATA GENERALIZADO PARA TODOS LOS ANALISADORES, TOMADO COMO REFERENCIA PARA LA PROGRAMACION DE CADA UNO.

Código de implementación de un analizador léxico:

Analizador css:

Analisador léxico para el lenguaje css

```
#analisador lexico para HTML
import re
class AnalisadorLexicoCss:
    def __init__(self):
        pass
    linea = 0
    columna = 0
    counter = 0
    Errores = []
    Comentarios = []
    Bitacora = []
    path = ''
    reservadas = ['color','font','size','background','margin','top','bottom','tex
t','align','position','hover','before','after','container','header','content',
                  'border','weight','padding','left','line','height','opacity',
'family','right','width','image','style','display','margin','float','clear','max'
,'min',
                  'px','em','vh','vw','in','cm','mm','pt','pc','url','content']
    signos = {"PUNTOCOMA":';', "LLAVEA":'{', "LLAVEC":'}', "ParA":'\(', "ParC":'\
)', "IGUAL":'=', "diagonal":'/', "dosPuntos":':', "asterisco":'\*'}
    signos2 = {"numeral":'#',"admiracion":'!',"porcentaje":'%',"pipe":'\|',"punto
":'\.',"comillasDobles":'"',"guionMedio":'-', "coma":',',"gionBajo":'_'}
    comentario = { "diagonalDoble":'/',"comillasDoblesxd":'"'}
    #EXPRESIONES REGULARES PARA IMPLEMENTACIÓN DE ANÁLISIS LÉXICO

    def inic(self,text):
```

```
        global linea, columna, counter, Errores, Comentario
        linea = 1
        columna = 1
        listaTokens = []
        counter= AnalisadorLexicoCss.counter
        while counter < len(text):
            if text[counter]=='u' and text[counter+1]=='r' and text[counter+2]=='
l':
                listaTokens.append(StateIdentifier(linea, columna, text, text[cou
nter]))
                counter += 4
                listaTokens.append(StateUrl(linea, columna, text, ''))
                counter += 1
                columna += 1
            elif re.search(r"[A-Za-z']", text[counter]): #CADENA
                listaTokens.append(StateIdentifier(linea, columna, text, text[cou
nter]))
            elif re.search(r"[0-9]", text[counter]): #NUMERO
                listaTokens.append(StateNumber(linea, columna, text, text[counter
]))
            elif re.search(r"[\n]", text[counter]):#SALTO DE LINEA
                counter += 1
                linea += 1
                columna = 1
            elif re.search(r"[ \t]", text[counter]):#ESPACIOS Y TABULACIONES
                counter += 1
                columna += 1
            elif text[counter]=='/' and text[counter+1]=='*':
                counter += 1
                AnalisadorLexicoCss.Comentarios.append(StateComent(linea, columna
, text, ''))
                counter += 1
                columna += 1
            else:
                #SIGNOS
                isSign = False
                for clave in AnalisadorLexicoCss.signos:
                    valor = AnalisadorLexicoCss.signos[clave]
                    if re.search(valor, text[counter]):
                        AnalisadorLexicoCss.Bitacora.append(['ESTADO SIGNO ',line
a, columna, text[counter]]) #aca llenamos el vector de bitacora
                        listaTokens.append([linea, columna, clave, valor.replace(
'\\','')])
                        counter += 1
                        columna += 1
```

```
                        isSign = True
                        break
                    else:
                        for clave2 in AnalisadorLexicoCss.signos2:
                            valor2 = AnalisadorLexicoCss.signos2[clave2]
                            if re.search(valor2, text[counter]):
                                AnalisadorLexicoCss.Bitacora.append(['ESTADO SIGN
O ',linea, columna,text[counter]]) #aca llenamos el vector de bitacora
                                listaTokens.append([linea, columna, clave2, valor
2.replace('\\','')])
                                counter += 1
                                columna += 1
                                isSign = True
                                break

                if not isSign:
                    columna += 1
                    AnalisadorLexicoCss.Errores.append([linea, columna, text[coun
ter]])
                    counter += 1
        return listaTokens

#[linea, columna, tipo, valor]

def StateIdentifier(line, column, text, word):
    global counter, columna
    counter += 1
    columna += 1
    if counter < len(text):
        if re.search(r"[a-zA-Z_0-9]", text[counter]):#CADENA
            return StateIdentifier(line, column, text, word + text[counter])
        else:
            AnalisadorLexicoCss.Bitacora.append(['ESTADO IDENTIFICADOR ',line, co
lumn, word]) #aca llenamos el vector de bitacora
            return [line, column, 'Cadena', word]
            #agregar automata de identificador en el arbol, con el valor
    else:
        AnalisadorLexicoCss.Bitacora.append(['ESTADO IDENTIFICADOR ',line, column
, word])#aca llenamos el vector de bitacora
        return [line, column, 'cadena', word]

def StateNumber(line, column, text, word):
    global counter, columna
    counter += 1
    columna += 1
```

```
    if counter < len(text):
        if re.search(r"[0-9]", text[counter]):#ENTERO
            return StateNumber(line, column, text, word + text[counter])
        elif re.search(r"\.", text[counter]):#DECIMAL
            return StateDecimal(line, column, text, word + text[counter])
        else:
            AnalisadorLexicoCss.Bitacora.append(['ESTADO NUMERO ENTERO',line, col
umn, word])#aca llenamos el vector de biacora
            return [line, column, 'integer', word]
            #agregar automata de numero en el arbol, con el valor
    else:
        AnalisadorLexicoCss.Bitacora.append(['ESTADO NUMERO ENTERO',line, column,
 word])#aca llenamos el vector de biacora
        return [line, column, 'integer', word]

def StateDecimal(line, column, text, word):
    global counter, columna
    counter += 1
    columna += 1
    if counter < len(text):
        if re.search(r"[0-9]", text[counter]):#DECIMAL
            return StateDecimal(line, column, text, word + text[counter])
        else:
            AnalisadorLexicoCss.Bitacora.append(['ESTADO NUMERO DECIMAL ',line, c
olumn, word])#aca llenamos el vector de bitacora
            return [line, column, 'decimal', word]
            #agregar automata de decimal en el arbol, con el valor
    else:
        AnalisadorLexicoCss.Bitacora.append(['ESTADO NUMERO DECIMAL ',line, colum
n, word])#aca llenamos el vector de bitacora
        return [line, column, 'decimal', word]

#si en el comentario no viene el cierre muere el programa, y si viene de ultimo
#tiene que venir al menos un espacio para la comprovacion que ahi termina el come
ntario
def StateComent (line,column, text, word ):
    global counter, columna, linea
    pattern = '/\*'
    counter += 1
    # columna += 1
    if counter < len(text):
        for match in re.findall (pattern, text):
            clave = text[counter]
            clave2 = text[counter+1]
```

```
            if clave != '*' or clave2 != '/': # el delimitador para el comentario
 es solo el * (no el conjunto de */)
                if clave =="\n":
                    linea+=1
                    return StateComent(line, column, text, word + ' ')

                else:
                    return StateComent(line, column, text, word + text[counter])
            else:
                AnalisadorLexicoCss.Bitacora.append(['ESTADO COMENTARIO ',line, c
olumn, word]) #aca llenamos el vector de bitacora
                return [line,column,'comentario',word]

def StateUrl (line,column, text, word ):
    global counter, columna, linea
    pattern = 'url'
    counter += 1
    columna += 1
    if counter < len(text):
        for match in re.findall (pattern, text):
            clave = text[counter]
            clave2 = text[counter+1]
            if clave != '"' or clave2 != ')': # el delimitador para el comentario
 es solo el * (no el conjunto de */)
                if clave =="\n":
                    linea+=1
                    return StateUrl(line, column, text, word + ' ')

                else:
                    return StateUrl(line, column, text, word + text[counter])
            else:
                AnalisadorLexicoCss.Bitacora.append(['ESTADO URL ',line, column,
word])#aca llenamos el vector de bitacora
                return [line,column,'URL',word]

def Reserved(TokenList):
    for token in TokenList:
        if token[2] == 'identificador':
            for reservada in AnalisadorLexicoCss.reservadas:
                palabra = r"^" + reservada + "$"
                if re.match(palabra, token[3], re.IGNORECASE):
                    token[2] = 'reservada'
                    break

def analyze(self, entrada):
```

```
        tokens = self.inic(entrada)
        return tokens



# nombre='entrada3.css'
# entrada = open(nombre,"r",encoding="utf-8")
# contenido = entrada.read()
# print(contenido)
# hola= AnalisadorLexicoCss()
# tokens = AnalisadorLexicoCss().inic(contenido)
# Reserved(tokens)
# for token in tokens:
#     print(token)
# print('ERRORES')
# for error in AnalisadorLexicoCss.Errores:
#     print(error)
# print ('COMENTARIOS')
# for coment in AnalisadorLexicoCss.Comentarios:
#     print(coment)
# print ('PATH')
# AnalisadorLexicoCss.path = AnalisadorLexicoCss.Comentarios[0]
# print (AnalisadorLexicoCss.path[3])
# print ('BITACORA')
# for bita in AnalisadorLexicoCss.Bitacora:
#     print (bita)
```