MANUAL TECNICO

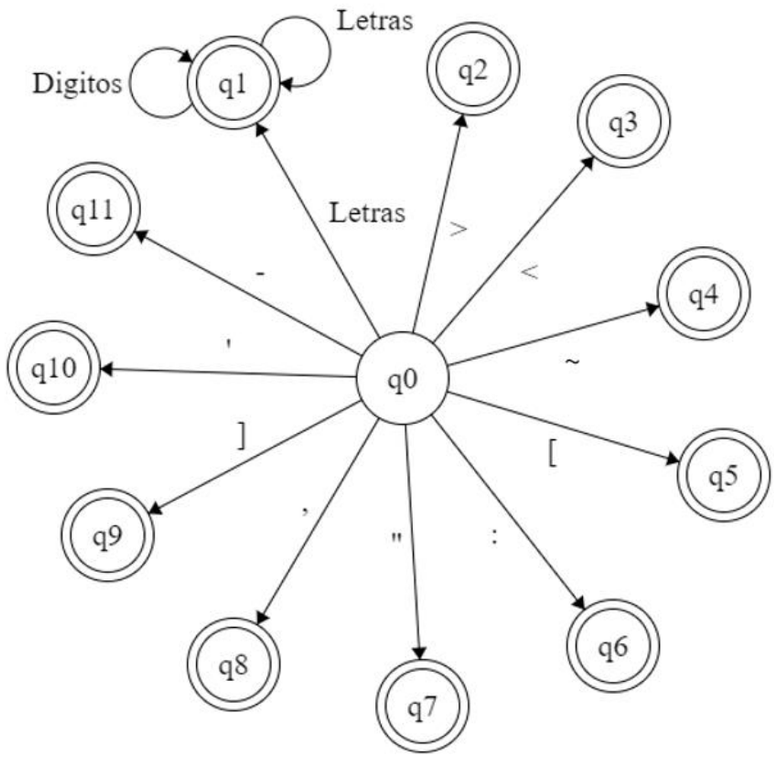
PROYECTO #1

MAX RODRIGO DURÁN CANTEO
RA: 201902219
UNIVERSIDAD SAN CARLOS DE GUATEMALA
INGENIRIA EN CIENCIAS Y SISTEMAS

INDICE

LECTURA

Se utilizo un solo AFD para leer el lenguaje form, como se ve en el diagrama:

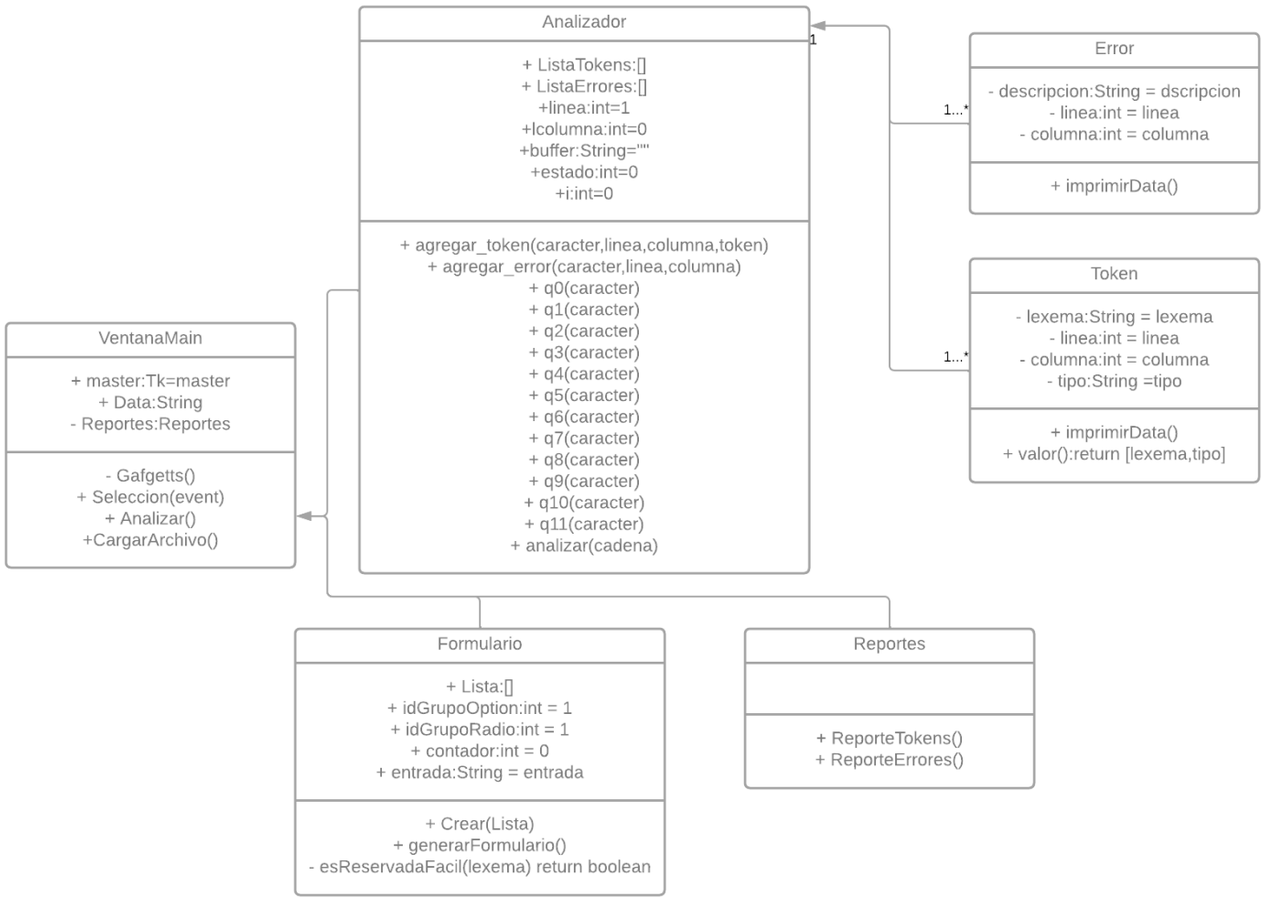


Creando once estados los cuales validaran los tokens permitidos que puede tener nuestro lenguaje, cualquier otro símbolo o combinación que no este definida en el AFD será tomado como un error y se guardara en nuestra lista de errores. Para los tokens válidos, cada uno tiene su propio identificador mostrado de la siguiente manera:

TOKEN	IDENTIFICADOR
valor	reservada_valor
tipo	reservada_tipo
fondo	reservada_fondo
nombre	reservada_nombre
valores	reservada_valores
evento	reservada_evento
Asd123	palabra
>	mayorQue
<	menorQue
~	Virgulilla
[	corcheteDerecho
:	dosPuntos
“	comillas
,	coma
]	corcheteDerecho
‘	comilla
-	guion

FUNCIONAMIENTO

El funcionamiento se describe en el siguiente diagrama de clases:



DESCRIPCION DE METODOS UTILIZADOS

CLASE VENTANAMAIN

GADGETTS

Genera el fronted del programa usando la librearía tkinter.

```
def Gadgetts(self):
    self.miFrame=Frame()
    self.miFrame.pack()
    self.miFrame.config(bg="pale green")
    self.miFrame.config(width="900", height="700")
    label_CargarArchivo=Label(self.miFrame, text="Cargar archivo .form",bg="pale green",fg="black", font=("Arial",12)).place(x=20,y=10)

    Faux=Frame()
    Faux.place(x=20,y=80)

    scroll=Scrollbar(Faux)
    scroll.pack(side="right",fill="y")

    self.textCaja=Text(Faux, width="105", height="35.5", yscrollcommand=scroll.set)
    self.textCaja.pack(side="left")

    scroll.config(command=self.textCaja.yview)

    self.botonCargarArchivo = Button(self.miFrame,text="Cargar Archivo",bg="spring green",activebackground="lawn green",font=("Arial",12),command=self.CargarArchivo)
    self.botonCargarArchivo.place(x=20,y=40)

    self.botonAnalizar = Button(self.miFrame,text="Analizar",bg="spring green",activebackground="lawn green",font=("Arial",12),command=self.Analizar)
    self.botonAnalizar.place(x=20,y=660)

    self.combobox=Combobox(self.miFrame,values=["Reporte de tokens","Reporte de errores","Manual de Usuario","Manual Técnico"],font=("Arial",12),state="readonly")
    self.combobox.place(x=675,y=45)
    self.combobox.current(0)
    self.combobox.bind('<<ComboboxSelected>>', self.seleccion)
```

SELECCIÓN

Abre el tipo de reporte o manual que el usuario pida.

```
def seleccion(self,event):
    valor=self.combobox.get()
    if valor=="Manual Técnico":
        webbrowser.open_new_tab("file:///"+os.getcwd()+"/Manuales/Manual Tecnico.pdf")
    elif valor=="Manual de Usuario":
        webbrowser.open_new_tab("file:///"+os.getcwd()+"/Manuales/Manual de Usuario.pdf")
    elif valor=="Reporte de errores":
        if os.path.exists("./Reportes/ReporteErrores.html"):
            webbrowser.open_new_tab("file:///"+os.getcwd()+"/Reportes/ReporteErrores.html")
        else:
            showinfo(title='ERROR',message="Aun no se a generado el reporte.")
    elif valor=="Reporte de tokens":
        if os.path.exists("./Reportes/ReporteTokens.html"):
            webbrowser.open_new_tab("file:///"+os.getcwd()+"/Reportes/ReporteTokens.html")
        else:
            showinfo(title='ERROR',message="Aun no se a generado el reporte.")
```

CARGARARCHIVO

Crea la opción de cargar el archivo from al programa.

```
def CargarArchivo(self):
    while True:
        try:
            ruta=easygui.fileopenbox(title="Abre el archivo tipo .form")
            extension=os.path.splitext(ruta)
            if extension[1].upper()==".FORM":
                print("[CARGAR ARCHIVO]: La extension es correcta")
                try:
                    Archivo=open(ruta,"r",encoding='utf-8')
                    Contenido=Archivo.read()
                    Archivo.close()
                    try:
                        self.textCaja.delete("1.0",END)
                        self.textCaja.insert(END,Contenido)
                        print("[CARGAR ARCHIVO]: Carga completada")
                    except:
                        print("[ERROR-CARGAR ARCHIVO]: Ocurrio un error al llevar la informacion a self.textCaja")
                    break
                except:
                    print("[ERROR-CARGAR ARCHIVO]: Ocurrio un error al leer el archivo")
            else:
                print("[ERROR-CARGAR ARCHIVO]: Extencion incorrecta")
        except:
            print("[ERROR-CARGAR ARCHIVO]: Ocurrio un error al abrir el archivo")
```

ANALIZAR

Crea al analizador y analiza los datos enviando la información cargada, luego crea los formularios dándole la información de los tokens y por último crea los reportes de los tokens y de los errores.

```
def Analizar(self):
    Data=str(self.textCaja.get("1.0",END))
    if Data!="":
        self.Data=Data
        print("[ANALIZAR]: Analizando...")
        analizador=Analizador()
        analizador.analizar(self.Data)
        formulario=Formularios(self.Data)
        formulario.Crear(analizador.listaTokens)
        self.Reportes.ReporteTokens(analizador.listaTokens)
        self.Reportes.ReporteErrores(analizador.listaErrores)
    else:
        print("[ERROR-ANALIZAR]: El campo esta vacio")
```

CLASE ANALIZADOR

ANALIZAR

Lee el texto y va llevando cada elemento del texto por los estados del AFD.

```
def analizar(self, cadena):
    cadena = cadena + '$'
    self.listaErrores = []
    self.listaTokens = []
    self.i = 0
    while self.i<len(cadena):
        if self.estado==0:
            self.q0(cadena[self.i])
        elif self.estado == 1:
            self.q1(cadena[self.i])
        elif self.estado == 2:
            self.q2(cadena[self.i])
        elif self.estado == 3:
            self.q3(cadena[self.i])
        elif self.estado == 4:
            self.q4(cadena[self.i])
        elif self.estado == 5:
            self.q5(cadena[self.i])
        elif self.estado == 6:
            self.q6(cadena[self.i])
        elif self.estado == 7:
            self.q7(cadena[self.i])
        elif self.estado == 8:
            self.q8(cadena[self.i])
        elif self.estado == 9:
            self.q9(cadena[self.i])
        elif self.estado == 10:
            self.q10(cadena[self.i])
        elif self.estado == 11:
            self.q11(cadena[self.i])
        self.i += 1
```

ESTADOS

Reciben su parámetro del método analizar y si cumple crea un token, de lo contrario crea un error.

```
def q0(self,caracter:str):···

def q1(self,caracter:str):···

def q2(self,caracter:str):···

def q3(self,caracter:str):···

def q4(self,caracter:str):···

def q5(self,caracter:str):···

def q6(self,caracter:str):···

def q7(self,caracter:str):···

def q8(self,caracter:str):···

def q9(self,caracter:str):···

def q10(self,caracter:str):···

def q11(self,caracter:str):···
```

AGREGAR_TOKEN

Si se cumplen los parámetros del estado de aceptación se agrega el token

```
def agregar_token(self,caracter,linea,columna,token):
    self.listaTokens.append(Token(caracter,linea,columna,token))
    self.buffer = ''
```

AGREGAR_ERROR

Si no se cumplen con los parámetros del estado de aceptación se agrega la sentencia a la lista de errores.

```
def agregar_error(self,caracter,linea,columna):
    self.listaErrores.append(Error(caracter, linea, columna))
```

CLASE FORMULARIO

GENERARFORMULARIO

Crea el formulario en base de html y le agrega un script para el funcionamiento de los botones, mostrando la información en un alert.

CLASE REPORTES

REPORTETOKENS

Crea un html con la lista de tokens válidos.

REPORTEERRORES

Crea un html con la lista de tokens inválidos.