MANUAL TECNICO

Analizador Léxico

Programador: William Mazariegos

Área: Lenguajes Formales y de Programación

Lenguaje utilizado: Python

Contenido

Descripción del Programa

Es un analizador léxico que realiza operaciones aritméticas básicas a través de un archivo de entrada con una determinada sintaxis que ayuda a poder extraer la información necesaria, además se analiza cada carácter con que está con construido el documento de entrada, buscando errores en su redacción para mostrarlo al usuario.

```
<Tipo>
<Operacion=SUMA>
	<Numero> 4.5</Numero>
	<Numero> 5.32 </Numero>
</Operacion>
<Operacion=INVERSO>
	<Numero>10</Numero>
	<Numero>20</Numero>
</Operacion>
</Tipo>
<Texto>
Realizar las operaciones básicas de suma, resta,
multiplicación y división, así como operaciones
complejas.
</Texto>
<Funcion = ESCRIBIR>
	<Titulo> Operaciones </Titulo>
	<Descripcion> [TEXTO] </Descripcion>
	<Contenido> [TIPO] </Contenido>
</Funcion>
<Estilo>
	<Titulo Color=AZUL Tamanio=16/>
	<Descripcion Color=VERDE Tamanio=13/>
	<Contenido Color=GRIS Tamanio=12/>
</Estilo>
```

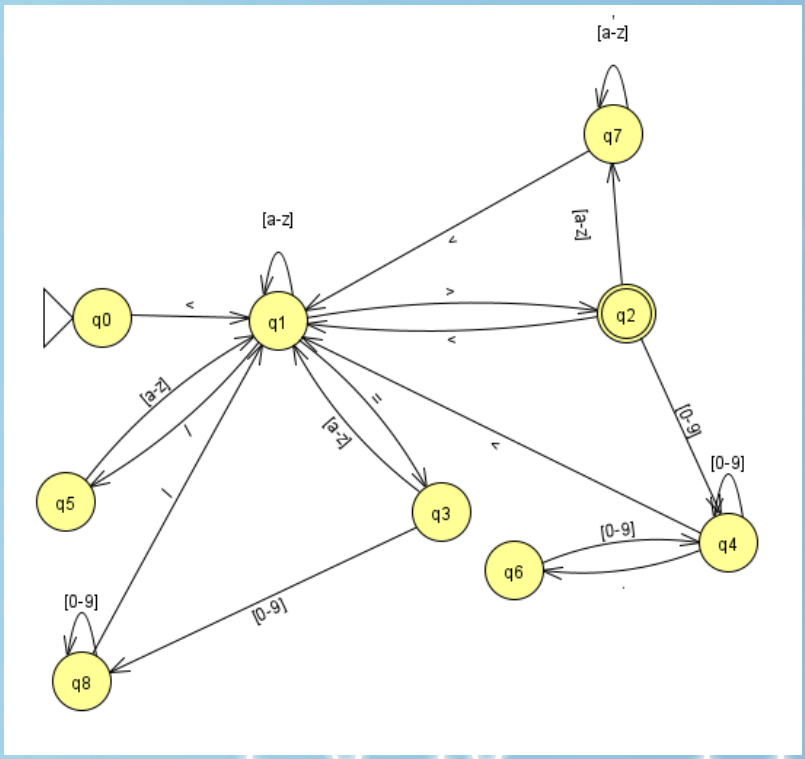
Ejemplo de archivo de entrada

En el cuadro anterior esta descrita la sintaxis que debería contener el archivo de entrada, para la verificación de los lexemas necesarios para que el programa funcione, se hizo un diagrama de un autómata finito determinista con el fin de conocer de forma grafica el comportamiento de los caracteres que conforman el archivo de entrada (“.txt”)

Definición del AFD

Para empezar con la graficar el autómata, se procedió a buscar todos los **tokens** que conforman el archivo, estos se describen en la siguiente tabla.

Token	Descripción
/	Diagonales
<>	Símbolos de mayor y menor que
[a-zA-Z]	Alfabeto
[0-9]	Números
λ	Espacios en blanco
\n	Saltos de línea
.	Puntos
[ ]	Corchetes



En la tabla anterior están todos los elementos de los que se compone el archivo de entrada. Ahora se procede a ver el comportamiento del archivo de entrada y luego se dibuja el autómata y tomando en cuenta las ocasiones en que un elemento puede cambiar de un estado a otro.

Vemos en la figura el diagrama de nuestro autómata finito determinista, este solo tiene un estado de aceptación que es el que nos va ayudar a asegurar que todas las etiquetas que

vienen en nuestro archivo de texto, sean debidamente cerradas con el símbolo “>”, y al no cerrar con este símbolo genere un error.

Código

Interfaz gráfica


```
from tkinter import *
from tkinter import scrolledtext
from tkinter import filedialog
from tkinter import messagebox
from tkinter.ttk import Treeview
```

Para la interfaz del programa se utilizó la librería **tkinter de Python**, esta ofrece elementos que ayudan con el GUI y facilitan el manejo en código.

Abrir archivos txt

Para poder escoger los archivos de entrada y configurarlo para que fueran solo archivos de texto, utilizaron las librerías **pathlib** y **filedialog**, esta ultima es una extensión de tkinter, ambas librerías ayudaron a filtrar el tipo de datos que se estaba seleccionando de forma gráfica.

```
def explorador():
    global archivo, contenidoruta, archivoextension
    archivo = filedialog.askopenfilename(filetypes=(
        ("Archivos de Texto", "*.txt"), ("Todos los ficheros", "*.*")), title="Seleccionar
    # initialdir="C:/", propieadad para inicar en una carpeta especifica
    extension = pathlib.Path(archivo)
    archivoextension = extension.name
    if str(extension.suffix) == ".txt":
        tokens_lista.clear()
        errores.clear()
        text_area.delete("0.0", END)
        labelruta.config(text=archivo)
        f = open(archivo, 'r')
        contenido = f.read()
        f.close()
        text_area.insert(END, contenido)
        contenidoruta = labelruta.cget("text")
        lbarchivocargado.config(background="white", text=str(
            extension.name), font=("Arial Black", 12))

    elif str(extension.suffix) != ".txt" and len(archivo) != 0:
        messagebox.showerror("", str(extension.name) +
                             " no es archivo valido")
```

Elementos de las ventanas

La librería **tkinter** nos ofrece una gran variedad de elementos que podemos agregar en nuestro código para ayudarle al usuario a utilizar de manera sencilla el programa. Entre estos elementos tenemos los botones, labels, cajas de texto, etc.

```
bt_saveas = Button(abrir_win, command=saveas)  # GUARDAR COMO
bt_saveas.config(image=img_saveas)
bt_saveas.place(x=850, y=160)
```

```
labelruta = Label(abrir_win, font=("Consolas", 12))
labelruta.place(x=120, y=20, width=650, height=40)
```

Leer archivos

El usuario el presionar el botón abrir lo lleva a una ventana donde se encuentra un text_area que le permite escribir y guardar ese documento que está escribiendo tal como si fuera un bloc de notas. Además, se encuentra un botón que le permite buscar un archivo .txt y cargarlo al sistema para que pueda ser analizado.

```python
def save():
    global contenidoruta
    contenidoruta = labelruta.cget("text")
    contenidocaja = text_area.get("1.0", 'end-1c')
    if len(contenidoruta) == 0:
        messagebox.showerror("", "No hay archivo para guardar.")
    else:
        modificado = open(contenidoruta, 'w')
        modificado.write(str(contenidocaja))
        modificado.close()
        messagebox.showinfo("", "Cambios realizados")

def saveas():
    global contenidoruta, archivoextension
    nuevoarchivo = filedialog.asksaveasfilename(title="Guardar como", initialdir="C:/", file
        ("Archivos de Texto", "*.txt"), ("Todos los ficheros", "*.*")), defaultextension="
    extension = pathlib.Path(nuevoarchivo)
    archivoextension = extension.name
    if len(nuevoarchivo) != 0:
        labelruta.config(text=nuevoarchivo)
        contenidocaja = text_area.get("1.0", 'end-1c')
        nuevo = open(nuevoarchivo, 'w')
        nuevo.write(str(contenidocaja))
```

Analizar archivos

Esta es la parte más extensa del código, porque en esta parte es donde se hicieron las validaciones que en base a nuestro diagrama AFD, se pudiera realizar los cambios de estado de todos los elementos que componen el archivo de entrada.

```python
    for char in mensaje:
        if char == '\n':
            row += 1
            col = 0
            continue
        elif char == '\t':
            col += 1
            continue
        elif char == ' ':
            col += 1
            continue
        if estado == 0:
            if char == '<':
                guardar_token(row, col, char)
                estado = 1
                estadoanterior = 0
                col += 1
                continue
            else:
                col += 1
                estado = 1
                guardarerror(row, col, char)
                continue
        elif estado == 1:
            c = char.translate(trans)
```

```python
    elif estado == 7:
        if estadoanterior == 2:
            auxfila = row
        c = char.translate(trans)
        if bool(letras.search(c.lower())) == True:
            col += 1
            estado = 7
            estadoanterior = 7
            auxtoken += char
            aux = col
        elif char == ' ' or char == '.' or char == ',' or char == ']':
            col += 1
            estado = 7
            estadoanterior = 7
            auxtoken += char
            aux = col
        elif char == '<':
            guardar_token(auxfila, aux, auxtoken)
            guardar_token(row, col, char)
            auxtoken = ""
            estado = 1
            estadoanterior = 7
            col = 0
        else:
```

Para guardar los **tokens** y los **errores** del archivo, se declararon listas que permitan manejar los datos de manera mas sencilla. Para eso se hizo un hicieron constructores que ayudaron a armar las listas y sus elementos.

```
class Token:
    def __init__(self,fila,columna,lexema):
        self.fila=fila
        self.columna=columna
        self.lexema=lexema

class Error:
    def __init__(self,No,lexema,tipo,columna,fila):
        self.fila=fila
        self.columna=columna
        self.lexema=lexema
        self.No=No
        self.tipo=tipo
```

```
def guardar_token(fila, columna, lexema):
    nuevotoken = Token(fila, columna, lexema)
    tokens_lista.append(nuevotoken)

def guardarerror(fila, columna, lexema):
    global conterrores
    conterrores += 1
    nuevotoken = Error(conterrores, lexema, "Error", columna, fila)
    errores.append(nuevotoken)
```

Tablas de Errores y de Tokens

Cuando el usuario presiona analizar, automáticamente se crean tablas para representar los errores y los tokens reconocidos del archivo. Para dibujar las tablas en la ventana se usó un módulo de la librería **tkinter** llamado **Treeview**.

```
# Tabla de errores
tabla = Treeview(analizar_win, columns=("c1", "c2"))
tabla.column("#0", width=60, anchor=CENTER)
tabla.column("c1", width=100, anchor=CENTER)
tabla.column("c2", width=100, anchor=CENTER)
tabla.heading("#0", text="Fila")
tabla.heading("c1", text="Columna")
tabla.heading("c2", text="Lexemna")
```

Generación de reportes HTML

Para generar los reportes, se usaron cadenas multilíneas para redactar la sintaxis del archivo HTML, habían partes que llevaban determinados datos del código y que incrustarlos en el archivo HTML, entonces se añadieron por partes de cadena a una cadena principal con la información que se necesitaba.

```
htmloperacion+="""
<html>
    <head>
        <title>Operaciones_202100123</title>
    </head>
    <style>
        .titulo{
            font-size: """+str(sizetitulo)+";"+"""
            color: """+str(colortitulo)+"""
        }
        .descripcion{
            font-size:"""+str(sizedescripcion)+";"+"""
            color: """+str(colordescripcion)+"""
        }
        .contenido{
            font-size:"""+str(sizecontenido)+";"+"""
            color: """+str(colorcontenido)+"""
        }
    </style>
    <body>
        <div class="titulo" >"""
```

```
        if len(raiz)==2:
            htmloperacion+="\t\t\t\tRAIZ: <br>\n"
            a=pow(raiz[0],1/raiz[1])
            cadope+="("+str(raiz[1])+")"+"√"+str(raiz[0])+"="+str(a)
            htmloperacion+="\t\t\t\t"+cadope+" <br><br>\n"
        cadope=""
        if len(modulos)>1:
            htmloperacion+="\t\t\t\tMODULO: <br>\n"
            aux=modulos[0]
            a=0
            cadope+=str(aux)+"%"
            for i in range(len(modulos)):
                if i>0:
                    a=aux % modulos[i]
                    aux=a
                    cadope+=str(modulos[i])+"%"
            cadope=cadope[:-1]
            cadope+="="+str(a)
            htmloperacion+="\t\t\t\t"+cadope+" <br><br>\n"
        cadope=""
        htmloperacion+="""\t\t\t</div>
    \t\t</body>
    \t</html>"""
```

Finalmente, cuando el usuario presiona el botón de generar el html, se ejecuta el código para escribir archivos y se le manda la cadena que se armó como contenido del archivo, y posteriormente se abre en el navegador de la computadora.

```
f = open('OPERACIONES_202100123.html','w')
f.write(htmloperacion)
f.close()
messagebox.showinfo("","Tabla de Operaciones generada con exito")
#time.sleep(2)
webbrowser.open_new_tab('OPERACIONES_202100123.html')
```