


UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS Y SISTEMAS
LENGUAJES FORMALES Y DE PROGRAMACION

PROYECTO FINAL

MANUAL TECNICO

Walter Gustavo Cotí Xalin
201700522

Guatemala, noviembre del 2020

Contenido

MENU

El inicio es un ciclo while, donde podrá elegirse una de las 4 opciones funcionales que se explican a detalle a continuación.

```
def menuBasilink():
    salir = False
    while salir == False:
        print("**************************************")
        print("*              Bienvenido            *")
        print("**************************************")
        print("1. Cargar Script")
        print("2. Manejo AFD")
        print("3. Pila Interactiva")
        print("4. Diagrama de Bloques de Codigo")
        print("5. Salir \n")
        try:
            opcMenu = int(input("---Seleccionar Una opcion---\n"))
        except:
            print("Solamente se permiten numeros")

        try:
            if opcMenu == 1:
                archivo = input('Nombre/direccion del archivo: ')
                contenido = openFile.getContenido(archivo)
            elif opcMenu == 2:
                Ana_Fila.analisis_linea(contenido)
                reportes.reportTokens()
            elif opcMenu == 3:
                #AutomataPila.iniciarAP()
                pass
            elif opcMenu == 4:
                print('opcion 4 man')
            elif opcMenu == 5:
                salir = True
            else:
                print("Las opciones unicamente son de 1 a 5")
```

CARGAR ARCHIVO

Esta opción imprime en pantalla un mensaje que pide el nombre o dirección del archivo .js, este es almacenado en una variable como texto para su posterior análisis.

En caso de no existir el archivo mostrará un mensaje de error.

```python
def getContenido(nameFile):
    try:
        with open (nameFile, 'r+') as data:
            contenido = data.read();
            return contenido
    except:
        print("no se encontro el archivo ->  " + nameFile)

def createFile(nameFile):
    pass
```

AFD

En la opción Manejo de AFD se tiene como parámetro una cadena, en este caso es el contenido del script .js, se comienza analizando carácter por carácter y concatenando cuando encuentre un símbolo o carácter que pertenezca al lenguaje, de no existir el símbolo se procede a guardarlo para posteriormente generar un reporte así como un reporte de tokens.

```python
def analisis_linea(contenido):
    estado = 0
    texto = ""
    numFila = 1
    numCol = 0
    lencontrr = len(contenido)
    for pos in range(len(contenido)):
        carcActual = contenido[pos]
        if contenido[pos] =="\n":
            numFila += 1
            numCol = 0
        else:
            numCol +=1
            if estado == 0:
                if contenido[pos].isalpha() or contenido[pos]== "_":
                    texto = texto + contenido[pos]
                elif contenido[pos].isdigit():
                    texto = texto + contenido[pos]
                else:
                    if contenido[pos] in tokens.lstTokens:
                        if texto:
                            if contenido[pos] == "(":

                                if texto.lower() in tokens.lstTokens:
                                    addToken(numFila,numCol-len(texto), texto,texto)
                                    texto=""
                                    estado = 3
                                else:
                                    addToken(numFila,numCol-len(texto),"identificador",texto)
                                    texto=""
                                    estado=3

                        if contenido[pos]=="=" and contenido[pos+1]==">":
                            addToken(numFila,numCol, "=>","=>")
                        else:
                            addToken(numFila,numCol, contenido[pos],contenido[pos])
                    elif contenido[pos]=="*" and contenido[pos-1]=="/":
                        addToken(numFila,numCol-1, "/*","/*")
                        estado = 6
```

```python
    elif estado == 1:#-----------------------------------IDENTIFICADOR--------------------------------------
        if contenido[pos].isdigit() :
            prueba = contenido[pos + 1]
            if contenido[pos + 1].isalpha() or contenido[pos + 1] == "_" :
                estado = 2
                texto = texto + contenido[pos]
                addToken(numFila,(numCol - len(texto)),"numero",texto)
                texto=""
            else:
                texto = texto + contenido[pos]
        elif contenido[pos] in tokens.lstTokens:
            addToken(numFila,numCol,contenido[pos],contenido[pos])
            estado = 2
        else:
            if contenido[pos].isspace():
                continue
            else:
                addErrores(numFila,numCol,contenido[pos])

    elif estado == 2:#-----------------------------------------------------------------
        if contenido[pos].isdigit() or contenido[pos].isalpha():
            texto = texto + contenido[pos]
        elif contenido[pos]== "_":
            texto = texto + contenido[pos]
        else:
            if contenido[pos] in tokens.lstTokens:
                if texto:
                    if texto.lower() in tokens.lstTokens:
                        addToken(numFila,numCol-len(texto), texto,texto)
                    elif texto.isnumeric():
                        addToken(numFila,numCol-len(texto), "numero",texto)
                    else:
                        addToken(numFila,numCol-len(texto), "identificador",texto)
                else:
                    estado = 3
                if contenido[pos] == ":" or contenido[pos] == ";":
                    estado = 0
                if contenido[pos] == ")":
```

El encontrar un símbolo perteneciente al lenguaje concateno todos los caracteres anteriores y verifico si es un numero, texto, booleano,etc.

Para almacenarlo en una lista de objetos token, que posee el token, valor, descripción, posición de fila y columna.

```python
elif estado == 4:#_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-
    if contenido[pos] == "\"":
        if texto:
            addToken(numFila,numCol-len(texto),"texto",texto)
        addToken(numFila,numCol,"\"","\"")
        texto = ""
        estado = 5
    else:
        texto = texto + contenido[pos]
elif estado == 5: #_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-
    if contenido[pos] in tokens.lstTokens:
        if contenido[pos] == ",":
            if texto:
                if texto.lower() in tokens.lstTokens:
                    addToken(numFila,numCol-len(texto), texto,texto)
                    texto =""
                elif texto.isnumeric():
                    addToken(numFila,numCol-len(texto), "numero",texto)
                    texto =""
                else:
                    addToken(numFila,numCol-len(texto), "identificador",texto)
                    texto =""
            addToken(numFila,numCol,",",",")
            texto = ""
            estado = 3
        elif contenido[pos] == ")":
            if texto:
                if texto.lower() in tokens.lstTokens:
                    addToken(numFila,numCol-len(texto), texto,texto)
                    texto =""
                elif texto.isnumeric():
                    addToken(numFila,numCol-len(texto), "numero",texto)
                    texto =""
                else:
                    addToken(numFila,numCol-len(texto), "identificador",texto)
                    texto =""
```

```
            elif contenido[pos] == ")":
                if texto:
                    if texto.lower() in tokens.lstTokens:
                        addToken(numFila,numCol-len(texto), texto,texto)
                        texto =""
                    elif texto.isnumeric():
                        addToken(numFila,numCol-len(texto), "numero",texto)
                        texto =""
                    else:
                        addToken(numFila,numCol-len(texto), "identificador",texto)
                        texto =""

                addToken(numFila,numCol,")",")")
                texto = ""
            elif contenido[pos] == ";":
                addToken(numFila,numCol,";",";")
                estado = 0
            elif contenido[pos] == "\"":
                addToken(numFila,numCol,contenido[pos],contenido[pos])
                estado = 4
        else:
            if contenido[pos].isspace():
                continue
            else:
                addErrores(numFila,numCol,contenido[pos])
    elif estado == 6:#------------------------------  comentarios----------------------
        if contenido[pos]=="*" and contenido[pos+1]=="/":
            if texto:
                addToken(numFila,numCol- len(texto),"comentario",texto)
            addToken(numFila,numCol, "*/","*/")
            texto = ""
            estado = 0
        else:
            texto = texto + contenido[pos]
```

Los métodos addToken y addError, listan los tokens y errores respectivamente en una lista para su posterior reporte

```
def addToken(fila, columna, preserv, valor):
    ntoken = obj_Token.obToken(tokens.lstTokens.get(preserv),valor)
    ntoken.setColum(columna)
    ntoken.setFila(fila)
    ntoken.setDescript(tokens.desc_Tokens.get(preserv))
    varGlobal.lst_Token_encontrados.append(ntoken)

def addErrores(fila, columna, texto):
    nError = obj_error.errorTo(fila,columna,texto)
    varGlobal.lst_Errores.append(nError)
```

REPORTAR

El reporte de tokens y errores se realiza de la misma forma, recorro una lista de objetos tokens, donde se obtiene y ordena la información para presentarlo en un archivo html.

Al finalizar la creación del mismo se ejecuta el html tanto del reporte de tokens como el de errores.

```
def reportTokens():
    print(str(len(varGlobal.lst_Token_encontrados)))
    lst_head = ['No.','Lexema','Token','Fila','Columna','Descripción']
    try:
        my_path = os.path.abspath(os.path.dirname(__file__))
        path = os.path.join(my_path, "R_Token.html")
        with  open(path, 'w+') as file_reporte:
            file_reporte.write(base_html.html_head("Report Tokens","Lista de Tokens"))
            file_reporte.write("<thead class=\"thead-dark\">")
            file_reporte.write("<tr>")
            for columna in lst_head:
                file_reporte.write("<th>"+ columna+ "</th>")
            file_reporte.write("</tr>")
            file_reporte.write("</thead")
            file_reporte.write("<tbody>")
            for cont in range(len(varGlobal.lst_Token_encontrados)):
                elem_Token = varGlobal.lst_Token_encontrados[cont]
                file_reporte.write("<tr>")
                file_reporte.write("<th>"+ str(cont+1) + "</th>")
                file_reporte.write("<td>"+ elem_Token.getLexema() + "</td>")
                file_reporte.write("<td>"+ elem_Token.getToken() + "</td>")
                file_reporte.write("<td>"+ str(elem_Token.getFila())+ "</td>")
                file_reporte.write("<td>"+ str(elem_Token.getCol()) + "</td>")
                file_reporte.write("<td>"+ elem_Token.getDescript() + "</td>")
                file_reporte.write("</tr>")
            file_reporte.write("</tbody>")
            file_reporte.write(base_html.final_hmtl)
        webbrowser.open_new_tab(path)
        print("Reporte de Tokens creado con Exito")
    except:
        print("Error al buscar algun dato")
```

AUTÓMATA DE PILA

La función iniciarAP inicia el análisis de pila del archivo ingresado, la función dametokens recorre la lista de objetos tokens y obtiene el token de cada objeto, retornando una lista con todos los tokens representado así la entrada del analizador sintactico.

```
def iniciarAP():
    Auto_Pila(dameTokens())
   # Auto_Pila(prueba)

def dameTokens():
    listaTokens=[]
    for obj_token in varGlobal.lst_Token_encontrados:
        listaTokens.append(obj_token.getToken())
    return listaTokens
```

Las primeras dos transiciones son únicamente para apilar el símbolo # y el Noterminal inicial.

```
def Auto_Pila(entrada):
    ent_ana = entrada
    salir = True
    estado = "i"
    pilaActual = []
    pila = []

    while salir== True:
        pilaActual= pila.copy()
        try:
            if estado == "i":#---------------------------------estado i--------------
                print(Fore.RED, 'PILA'+ "  |  " +Fore.YELLOW,'CADENA ENTRADA' + "  |  " +Fore.GREEN,'TRANSISION')
                #print('PILA' + "  |  " +'CADENA ENTRADA'+ "  |  " + 'TRANSISION')
                pila.append("#")
                printFormato(pilaActual,entrada,transisiones.tran[0])
               # print("|"+ pila + "  |  " + entrada[0] + "  |  " + transisiones.tran[0])
                estado = "p"

            elif estado == "p":#---------------------------------estado p--------------

                pila.append("SENT")
                printFormato(pilaActual,entrada,transisiones.tran[1])
                estado = "q"
```

En el estado q, es donde se llevan a cabo las transiciones, so comienza obteniendo la cima de la pila y el primer token de entrada.

```
    elif estado == "q":#---------------------------------estado q--------------

        cimaPila = pila[-1]
        if len(ent_ana)==0:
            TuplaTR= dameTuplaNoTerm(cimaPila,"E")
            printFormato(pilaActual,entrada,TuplaTR)

            if cimaPila == "#":
                printFormato(pilaActual,["----"],transisiones.tran[-1])
                pila.pop()
                estado = "r"

        else:
            tokenAct = entrada[0]
            if cimaPila in tokenAct:
                tplaTerminal = tupla_Terminal(cimaPila)
                printFormato(pilaActual,entrada,tplaTerminal)
                pila.pop()
                entrada.pop(0)
            else:
                try:
                    TuplaTRansision = dameTuplaNoTerm(cimaPila,tokenAct)
                    if type(TuplaTRansision) == None:
                        TuplaTRansisionconE = dameTuplaNoTerm(cimaPila,"E")
                        pila.pop()
                        printFormato(pilaActual,entrada,TuplaTRansisionconE)
                    else:
                        pila.pop()
                        for tken in reversed(TuplaTRansision[4]):
                            pila.append(tken)
                        printFormato(pilaActual,entrada,TuplaTRansision)
                    #print(pilaActual,entrada,TuplaTRansision)
                except:
                    print("transicion con Epsilon prro")
```

```
def tplaNoTerminal(No_terminal):
    for nterm in transisiones.tran:
        tmplist = nterm[4]
        if tmplist.__class__== list:
            if tmplist[0] == No_terminal:
                return nterm
        else:
            if tmplist == No_terminal:
                return nterm

def dameTuplaNoTerm(cimaPila, NTBuscar):
    bucle = True
    NTerm= tplaNoTerminal(NTBuscar)
    while bucle == True:
        if NTerm[2] == cimaPila:
            return NTerm
        else:
            NTerm = tplaNoTerminal(NTerm[2])

def tupla_Terminal(tkDesapilar):
    for nterm in transisiones.tran:
        tmplist = nterm[1]
        if tmplist == tkDesapilar:
            return nterm
```

La función print formato recibe únicamente listas, estas son concatenadas y después se procede a imprimir.

```
def printFormato(Pilaentrada, cadenaentrada, tran_entrada):
    txtPila =""
    txt_ent = ""
    txt_tran = ""
    produc = ""
    for itm in Pilaentrada:
        txtPila += itm+" "
    for tkns in cadenaentrada:
        txt_ent += tkns + " "
    for prod in tran_entrada[4]:
        produc += prod +" "
    txt_tran =  "(" + tran_entrada[0] + ", "+tran_entrada[1] + ", "+tran_entrada[2] + "; "+ tran_entrada[3] + ", "+ produc + ")"
    print(Fore.RED, txtPila + Fore.YELLOW,txt_ent + Fore.GREEN,txt_tran)
```

DIAGRAMA DE BLOQUES DE CÓDIGO