Universidad de San Carlos de Guatemala

Organización de Lenguajes y Compiladores 1

Ing. Manuel Castillo

Aux. Huriel

Manual Técnico

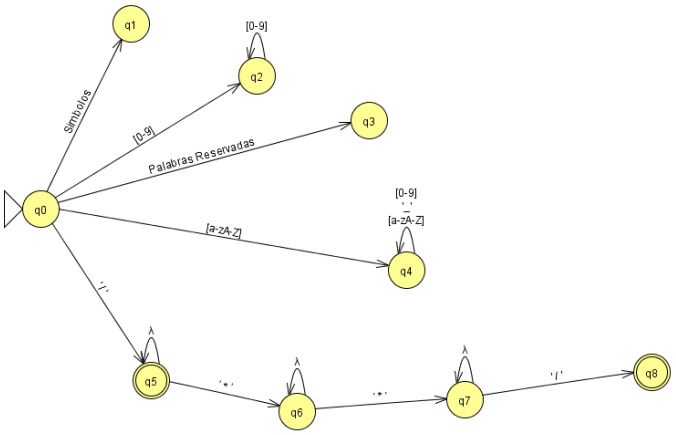
Elmer Gustavo Sánchez García

201801351

Fecha de entrega 16/09/2020

Autómata

Para el analizador léxico se utilizo el siguiente autómata que reconoce sintaxis para JavaScript, se analizó carácter por carácter para buscar caracteres que no pertenezcan al lenguaje.



Interfaz Grafica

Esta clase es la encargada de lanzar la aplicación de forma grafica, para que el usuario pueda interactuar con el programa, esta clase también posee los caja de texto para que el usuario pueda escribir sin ningún problema.

```
class Interface():
    def __init__(self,window):
        self.wind = window
        self.rutaHTML = ''
        self.wind.title('Analysis Application')
        self.wind.geometry("1000x700")
        self.wind.configure(bg = '#0A010B')


        self.textArea = CustomText(self.wind,width = 100,height = 25, bg ="#453B46", foreground = "#FFFFFF")
        self.txtConsola = Entry(self.wind, width = 10)
```

```
self.buttonOk = Button(self.wind,text ="JS" ,command = self.getText, bg = "#0A010B", highlightbackground = "#0A010B", highlightcolor= "#0A010
self.buttonOkCSS = Button(self.wind,text = "CSS",command = self.getTextCSS, bg = "#0A010B", highlightbackground = "#0A010B", highlightcolor=
self.buttonOkHTML = Button(self.wind,text = "HTML" ,command = self.getTextHTML, bg = "#0A010B", highlightbackground = "#0A010B", highlightcol
self.buttonOkSINT = Button(self.wind, text = "SINTACT.." ,command = self.getTextSINT, bg = "#0A010B", highlightbackground = "#0A010B", highli

#self.wind.iconphoto(False,self.icon)


self.lexema = ""
self.analizador = Analicis()
self.analizadorCSS = Anality_CSS()
self.analizadorHTML = Anality_HTML()
self.analizadorSINT = Sintactico()
```

Obtener Texto

Esto método se encarga obtener el texto que se encuentra en el cuadro de texto para enviar, a su respectivo análisis , dependiendo si es JavaScript, CSS, HTML o sintáctico.

```
def getTextCSS(self):
    self.txtConsola.delete("1.0","end")
    entrada = self.textArea.get("1.0",END)

    analizado = self.analizadorCSS.read_caracter(entrada)

    self.txtConsola.insert("2.0",analizado)
```

Abrir Archivo

Este método se encarga de abrir los obtener la ruta del archivo que uno desea abrir, para así plasmarlo en la consola del programa, y se pueda realizar el análisis correspondiente.

```
def open_File(self):
    try:
        ruta =  ""
        #root = Tk()
        filename =  filedialog.askopenfilename(initialdir = "/",title = "Select file",filetypes = (("JS files","*.js"),("CSS files","*.css"),(
        ruta = filename
        if ruta != "":
            self.write_consola(ruta)
            return ruta
        else:

            return None

    except IndexError as e:
        print(e)
```

Analizador CSS

Esta clase es la encargada de realizar todo el análisis que corresponda ha hojas de estilo en cascada (CSS) aquí se analiza la sintaxis, si viene un carácter que no pertenezca al lenguaje para ser reportado como un error.

```
def read_caracter(self,texto):
    self.bitacora = f"---------->Bitacora CSS <---------- [{self.getTime()}] \n"
    self.entrada = texto + '$'
    self.caracterActual = ''
    self.newEntrada = texto

    x = 0
    while x < len(self.entrada):
        self.caracterActual = self.entrada[x]
        #print(self.caracterActual)
        # qo -> q1(simbolos del lenguje)
        if self.caracterActual == '{':
            self.add_tokken(TipoCSS.LLABRE,'{',"")
        elif self.caracterActual == '}':
            self.add_tokken(TipoCSS.LLCIER,'}',"")
        elif self.caracterActual == '(':
            self.add_tokken(TipoCSS.PABRE,'(',"")
        elif self.caracterActual == ')':
            self.add_tokken(TipoCSS.PCIER,')',"")
        elif self.caracterActual == '"':
            self.add_tokken(TipoCSS.COMIDOBLE,'"',"")
        elif self.caracterActual == ',':
            self.add_tokken(TipoCSS.COMA,',',"")
        elif self.caracterActual == ':':
            self.add_tokken(TipoCSS.DPUNTOS,':',"")
        elif self.caracterActual == ';':
            self.add_tokken(TipoCSS.PCOMA,';',"")

        elif self.caracterActual.isnumeric():
            #-------------------Agregar bitacora---------------------------------------
            _trasBitacora = f"[q0 -> q2;{self.caracterActual}] - [{self.getHora()}] \n"
```

Estado q3(CSS)

Este es el estado encargado de analizar el carácter si es letra y que no venga un carácter que no pertenece al lenguaje en esa parte de la cadena.

```
# ----------------->ESTADO Q3 <-------------------------------------
# Letra
def q3(self,actual,fin):
    c = ''
    while actual < fin:
        c = self.entrada[actual]
        # q3 -> q3 con letra (letra)
        if c.isalpha():
            self.lexema += c
            _trasBitacora = f"[q3 -> q3;{c}] - [{self.getHora()}] \n"
            self.bitacora += _trasBitacora

            if (actual + 1 == fin):
                self.add_tokken(TipoCSS.VALOR,self.lexema,"blue")
                _trasBitacora = f"[Acepatado ;{self.lexema}] - [{self.getHora()}] \n"
                self.bitacora += _trasBitacora
        else:
            #Caracter No Reconocido
            self.add_error(actual,self.linea,self.columna,c)
        actual +=1
        self.columna +=1
```

Retorno del tamaño del lexema

Este método es el encargado de retornar el tamaño del lexema que se analizara, dependiendo cual es su primer carácter si es numero o letra o pertenece a una palabra reservada de lenguaje.

```
def get_size_lexema_asterisco(self,incio):
    longitud = 0
    for j in range(incio,len(self.entrada) - 1):
        if self.entrada[j] == "\n":
            self.linea += 1
            self.columna = 1
            #print("Salto de linea get_size_lexema")
        if self.entrada[j] == " " or self.entrada[j] == "," or self.entrada[j] == ';' or self.entrada[j] == ':' or self.entrada[j] == "\n"
            break
        longitud += 1
    return longitud
```

Clase Reporte

Esta clase es la encargada de generar todos los reportes utilizados como los reportes para JavaScript, HTML y CSS se encarga de generar los reportes en graphviz como los de html y la recuperación de errores

```
class Report():
    rutaJs = ''
    text_RecuperacionError = ''
    ruta_generica = ''
    def __init__(self):
        self.rutaJs = ''
        self.ruta_generica = ''
        self.text_RecuperacionError = ''
```

Retornar la ruta

Este método es el encargado de buscar la ruta dentro de los token de tipo comentario, después de encontrar la el comentario realizara un búsqueda de la ruta concreta y la devolverá para su uso

```
def get_pathComenatrio(self):
    ruta = ''
    for valor in self.lista_Tokens:
        if TipoCSS.COMENTARIO == valor.getTipoToken():
            tokenValor =  valor.getValorToken()
            if(tokenValor.find("PATHW")) != -1:
                if(tokenValor.find("c:") != -1):
                    ruta = tokenValor[tokenValor.find('c:'):tokenValor.find('*/')].strip()
                    break
                elif (tokenValor.find("C:") != -1):
                    ruta = tokenValor[tokenValor.find('C:'):tokenValor.find('*/')].strip()
                    break


    return ruta
```

Retornar la Hora

Este método es el encargado de retorno la hora actual, para su uso en la bitácora de css

```
def getHora(self):
    time = ""
    minuto = self.now.minute
    if (minuto <= 9):
        minuto = f"0{self.now.minute}"
        time = f"{self.now.hour}:{minuto}.{self.now.second} "
    else:
        time = f"{self.now.hour}:{self.now.minute}.{self.now.second} "

    return time
```

Generar Graphviz

Este método se encarga de generar el texto de graphviz para luego escribir en un archivo este texto, este método funciona recorriendo un diccionario con los estados y con qué carácter va cada estado, funciona en dos bloques, retorna el texto completo.

```
def genenarte_Graphivz(self,dic_tranciones):
    bloqueUno = "R[shape=point] \n"
    bloqueDos = ' R -> q0 [arrowhead="dot",color="#832561"] \n'
    caracter = ''
    for key,values in dic_tranciones.items():
        bloqueUno += '%s [label = "%s"]     \n'%(key,key)

    for key,values in dic_tranciones.items():
        for valor in values:
            transicion = valor.split(",")
            caracter = transicion[0]
            if caracter.find("\\") != -1:
                caracter = caracter.replace("\\",'/')
                #print(f"entra al primer if:{caracter}")
            if caracter.find('"') != -1:
                caracter = caracter.replace('"',"'")
                #print(f"entra al 2 if:{caracter}")


            bloqueDos += ' %s -> %s [arrowhead="vee",color="#832561", label="%s"] \n'%(key,transicion[1],caracter)

    texto = 'digraph G{ \n rankdir=LR; \n node[shape="circle",fontcolor="#832561",color="#B965AE"];\n %s \n %s \n}'%(bloqueUno,bloqueDos)
    return texto
```

Reporte HTML

Este método es el encargado de generar el reporte de HTML, cual consta de tres bloques el principal y dos secundarios, el listado de erros se va concatenado el segundo bloque, este posee la estructura de un documento de HTML

```
def reporteHTMLCSS(self,texto,lista_Error):
    x = 1
    newTexto = self.solunionarError(texto,lista_Error)
    self.text_RecuperacionError = newTexto
    head = '<head> \n \t<meta charset="UTF-8">\n \t<meta name="viewport" content="width=device-width, initial-scale=1.0">\n \t<title>Report</titl
    bloqueUno = '\t<table class="table table-hover">\n \t \t <tr> \n \t \t \t <td scope="col"><strong>No.</strong></td>\n \t \t \t <td scope="col
    bloqueDos = ''
    for error in lista_Error:
        caracter = f"El carcter '{error.getCaracter()}' no pertenece al lenguaje."
        bloqueUno += '\t \t<tr>\n \t \t \t<td scope="col">%s</td>\n \t \t \t<td>%s</td>\n \t \t \t<td>%s</td> \n \t \t </tr>\n \n'%(x,error.getLi
        x += 1
    bloqueUno += '\t</table>\n'
    bloqueDos += '\t<h2>Recuperacion de Error</h2>\n \t <p>%s</p>\n </br>\n </br>\n </br>\n </br> \n \t <footer> \n \t \t <p><strong>Autl
    bloquePrincipal = '<!DOCTYPE html>\n <html lang="en">\n %s \n <body>\n <h1>Listado Errores lexicos</h1>\n %s \n %s \n </body>\n </html>\n'%(l

    #print(bloquePrincipal)
    return bloquePrincipal
```