



TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE MEXICALI

Materia:

Lenguajes y autómatas I

Unidad 2:

Expresiones regulares

Trabajo:

Analizador Léxico

Alumno:

Cardenas Flores Andree Jared

No. De control:

20490691

Docente:

Carlos Alberto López Castellanos

MEXICALI, B.C. A 12 DE MARZO DE 2023

Introducción:

En este trabajo veremos cómo hacer y utilizar un analizador léxico, pero para empezar hay que conocer que es y cómo funciona.

El analizador léxico es una herramienta de un compilador que nos ayuda a leer los caracteres de entrada para formar componentes y así poder identificarlos y pasar la información a analizadores sintáctico.

Los componentes léxicos representan:

- Palabras reservadas: if, while, do ect.
- Operadores. +=*/== > <!=.
- Constantes numéricas.
- Constantes de caracteres.

¿Cuál es su función?

Construye elementos léxicos llamados patrones que serán utilizados posteriormente por un analizador sintáctico. Un patrón es una pareja ordenada compuesta por un token y un lexema.

Un lexema es la secuencia de caracteres que coinciden con tokens.

Un analizador léxico aísla el analizador sintáctico de la representación de lexemas de los componentes léxicos.

El analizador léxico opera bajo petición del analizador sintáctico devolviendo un componente léxico conforme el analizador sintáctico lo va necesitando para avanzar en la gramática. Los componentes léxicos son los símbolos terminales de la gramática.

Suele implementarse como una subrutina del analizador sintáctico. Cuando recibe la orden obtén el siguiente componente léxico, el analizador léxico lee los caracteres de entrada hasta identificar el siguiente componente léxico.

Otras Funciones:

- Manejo del fichero de entrada del programa fuente: abrirlo, leer sus caracteres, cerrarlo y gestionar posibles errores de lectura.
- Eliminar comentarios, espacios en blanco, tabuladores y saltos de línea (caracteres no válidos para formar un token).
- Inclusión de ficheros: # include...
- La expansión de macros y funciones in line: # define...
- Contabilizar el número de líneas y columnas para emitir mensajes de error.
- Reconocimiento y ejecución de las directivas de compilación (por ejemplo, para depurar u optimizar el código fuente).

Salida del analizador léxico

A la salida del analizador léxico es un conjunto de tokens.

Ejemplo:

- Identificadores.
- Enteros.
- Palabras reservadas.
- Espacios en blanco.
- Paréntesis.

Definiciones.

Tokens:

Símbolos terminales de una gramática

- Identificadores, palabras reservadas, operadores...
- Varios signos pueden formar el mismo token

Atributos:

Información adicional que tiene el token, de utilidad para el análisis sintáctico y semántico.

Componentes léxicos (tokens):

Unidad mínima de información que significa algo a la hora de compilar; concepto de palabra; las fases de un lenguaje constan de cadenas de componentes léxicos.

Lexema:

Una secuencia de caracteres de entrada que comprenden un solo componente léxico se llama lexema; cadena de caracteres que extrae el componente abstracto del componente léxico.

Patrón:

Descripción de la forma que han de tomar los lexemas para ajustarse a un componente léxico.

Desarrollo:

Objetivo General:

En esta práctica se pretende construir un analizador léxico, el cual identifica cada símbolo de la cadena de caracteres que se pretenda leer desde un archivo ya hecho, informando al usuario que token es y cuales no son válidos o no reconocidos.

Objetivo Particular:

Para lograr esto lo haremos mediante el uso de expresiones regulares establecidas que ayudan a la comparación de cadenas, logrando con esto la separación e identificación de caracteres válidos y no válidos.

El programa pide al usuario que introduzca un archivo de texto ya con el texto a analizar y posteriormente el programa comienza a comparar las cadenas con las expresiones y esta las imprime en la consola. Este analizador está programado en lenguaje Java.

Clases del programa:

Clase Token:

La clase Token es un componente importante para el analizador léxico este nos ayuda a identificar y validar tokens en un programa de entrada utilizando expresiones regulares.

Esto lo hace utilizando el método de validar, este acepta un argumento cadena que representa el texto que va a validar, esto lo analiza con el patrón del token y si este coincide devuelve el nombre del token y si no coincide devuelve "Error"

También dentro de esta misma clase tenemos un constructor que acepta dos parámetros, el nombre del token y el patrón que describe la expresión regular.

Clase Nodo:

Esta clase se utiliza para construir una lista enlazada de tokens en el analizador léxico. Cada nodo contiene un token y un enlace al siguiente nodo en la lista. La clase tiene un método validar que utiliza la clase token para validar una cadena de entrada y devolver el nombre del token.

Esto lo hace mediante un método constructor que acepta los dos parámetros el nombre del token y la expresión regular que describe el patrón del token. El constructor crea un nuevo objeto Token que utiliza los parámetros y los almacena en la variable en dato

También tenemos el get y set de siguiente, el método getSig devuelve el siguiente nodo de la lista y el método setSig establece el siguiente nodo de la lista.

Además, tiene un método validar este acepta el argumento de cadena que representa el texto a validar, esta llama al método validar de la clase token utilizando el objeto dato y la cadena. Esto valida la cadena y devuelve el nombre del token correspondiente. Si la validación falla, se lanza una excepción.

Clase Lista:

Esta clase se utiliza para construir una lista enlazada de tokens en un analizador léxico. La lista enlazada se utiliza para almacenar los tokens encontrados en un programa de entrada para su análisis. La clase tiene métodos para agregar tokens a la lista, validar una cadena y leer una lista de tokens y patrones desde un archivo.

Esta se integra de un método `empty` de tipo booleano que devuelve si la lista está vacía o falso si tiene elementos.

También tiene el método `vaciarLista` que vacía la lista.

Además, tiene un método `insertar` este acepta dos argumentos que son el nombre del token y el patrón. Este crea un nuevo nodo utilizando los argumentos y lo inserta al final de la lista, si la lista está vacía, tanto la raíz como el último se establecen en el nuevo nodo.

Asimismo tenemos el método `validarCadena` este acepta un argumento que son la cadena y este representa el texto a validar, este método recorre cada nodo en la lista en orden y llama al método `validar` de cada nodo utilizando la cadena. Si la cadena es válida según el patrón de un token este devuelve el nombre de ese token. Si la cadena no coincide con ningún patrón de token, este devuelve "Error".

Igualmente tenemos el método `LlenarLista` que acepta un argumento que es el archivo y este representa el nombre del archivo que contiene la lista de tokens. Este método utiliza la clase `archivo` para leer cada línea del archivo y llama al método `insertar` de la misma clase para agregar el nombre del token y el patrón a la lista.

Clase Archivo:

Esta clase permite leer un archivo de texto y separar las palabras por espacios

Esta clase se integra por un constructor de la clase y un método llamado `SepararPalabras`.

El constructor de la clase toma como parámetro el nombre del archivo que lee y crea un objeto `FileReader` y un objeto `BufferedReader` para leer el archivo en caso de no encontrar el archivo este imprime "Error al leer el archivo".

El método `SepararPalabras` se encarga de leer el archivo línea por línea y separa las palabras por espacios en cada línea. Este método verifica si la línea comienza con "#" o está vacía. Si es así, se salta la línea. De lo contrario, se usa el método `"split"` de Java para dividir la línea en palabras separadas por espacios en blanco. La expresión regular `"\s+(?=[^"]"[^"]*)"[^"]$"` se utiliza para ignorar los espacios dentro de las comillas dobles. Si el archivo se lee correctamente, se devuelve un array `String` con las palabras separadas. Si el archivo está vacío o no se encuentra se devuelve un array `String` con el mensaje de "Error"

Clase Programa:

Esta es la clase principal del programa que analiza un archivo de código en busca de tokens utilizando una lista de tokens predefinidos y luego imprime los tokens detectados.

Esta clase utiliza las clases Lista y Archivo para analizar un archivo de código y detectar los tokens utilizados en él.

En la función main, se crea un objeto de la clase lista y luego se utiliza el método Llenarlista para llenarla con tokens desde un archivo llamado TablaTokens.txt, Luego se crea un objeto de la clase Archivo con el nombre del archivo a analizar.

Se utiliza un bucle “do-while” para leer cada línea del archivo y separar las palabras de la línea utilizando el método SepararPalabras de la clase Archivo. Cuando detecta que el archivo a terminado el bucle se detiene e imprime “Fin del archivo”.

Para cada palabra separada, se utiliza el método validarCadena de la clase Lista para determinar si es un token valido. El nombre se imprime en la consola.

Análisis de los resultados:

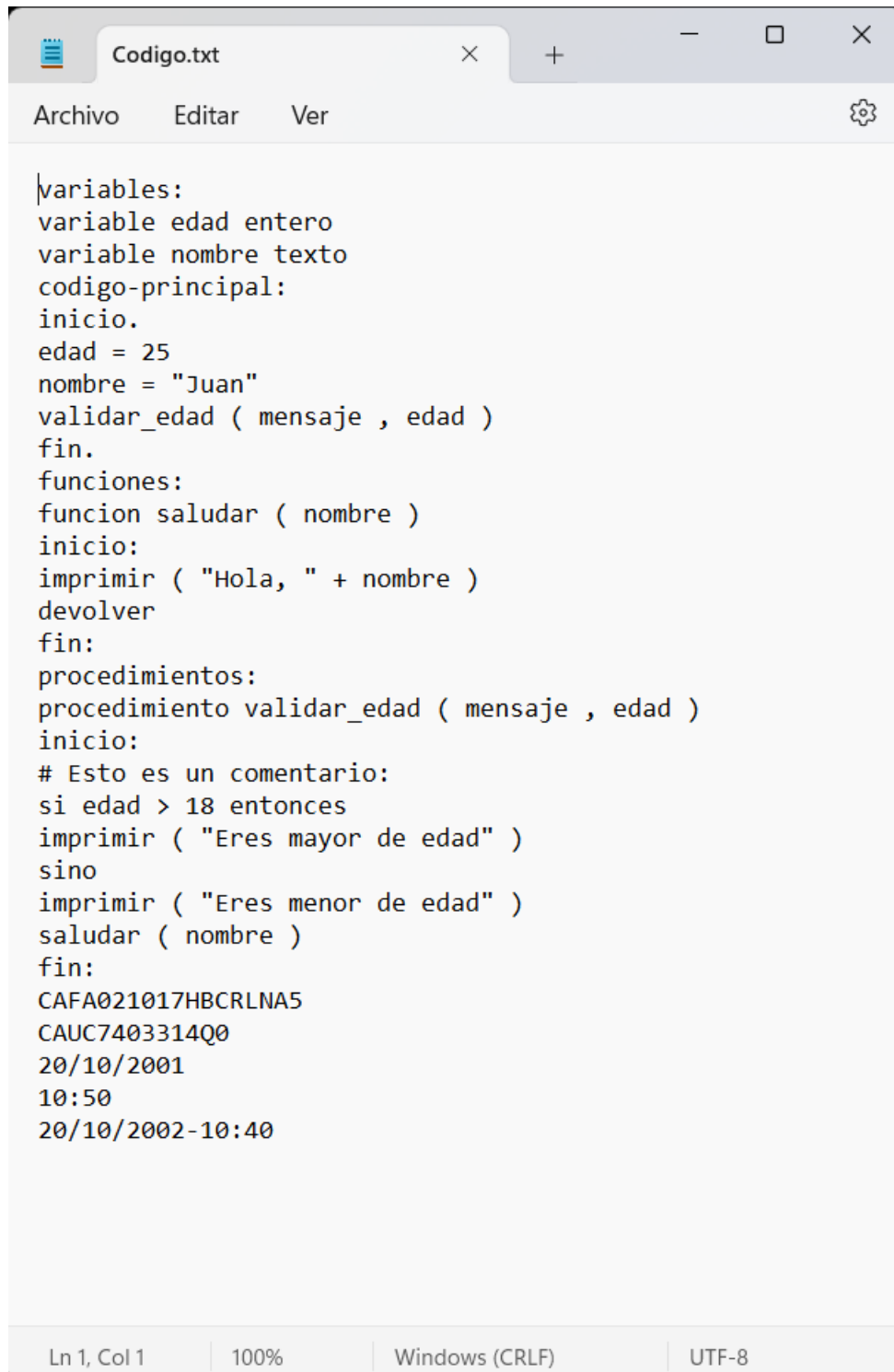
Para el análisis de los resultados pondré imágenes de los 2 archivos de texto que uno es la tabla de tokens y el otro es un código que fue proporcionado por el profesor, además también pondré el resultado en consola del programa una vez ejecutado en donde se mirara el nombre de los tokens que detecte que están adentro del código.

```

TknVariables ^variables:$
TknVariable ^variable$
TknEntero ^entero$
TknDecimal ^decimal$
TknSimbolo ^simbolo$
TknTexto ^texto$
TknLogico ^logico$
TknFechaHora ^fecha-hora$
TknFecha ^fecha$
TknHora ^hora$
TknFunciones ^funciones$
TknProcedimientos ^procedimientos$
TknFuncion ^función$
TknDevolver ^devolver$
TknComo ^como$
TknProcedimiento ^procedimiento$
TknInicio ^inicio$
TknFin ^fin$
TknSiEntoncesSino ^si-entonces-sino$
TknPara ^para$
TknMientras ^mientras$
TknRomper ^romper$
TknSuma ^[+]$
TknResta ^[-]$
TknMulti ^[*]$
TknDivision ^[/]$
TknModulo ^[%]$
TknIncremento ^[++]$
TknDecremento ^[--]$
TknAsignacion ^[=]$
TknAsignacionConSuma ^[+=]$
TknAsignacionConResta ^[-=]$
TknAsignacionConMulti ^[*=]$
TknAsignacionConDivision ^[/=]$
TknAsigancionConModulo ^%=$
TknIgualdad ^==$
TknDesigualdad ^!=$
TknMayorQue ^>$
TknMenorQue ^<$
TknMayorOIgualQue ^>=$
TknMenorOIgualQue ^<=$
TknY ^_Y_$
TknO ^_O_$
TknNegacion ^~$
TknTexto ^texto$
TknCodigoPrincipal ^codigo-principal:$
TknInicio ^inicio.$
TknPunto ^[.]$
TknNumero ^-[0-9]+(\\.[0-9]+)?$
TknFin ^fin.$
TknFunciones ^funciones:$
TknInicio: ^inicio:$
TknProcedimientos ^procedimientos:$
TknComentario ^[#]$
TknSi ^si$
TknEntonces ^entonces$
TknParentesisAbre ^[(]$
TknParentesisCierra ^[)]$
TknMensaje ^mensaje$
TknComa ^[,]$
TknImprimir ^imprimir$
TknSino ^sino$
TknCurp ^[A-Z]{1}[AEIOU]{1}[A-Z]{2}[0-9]{2}(0[1-9]|1[0-2])(0[1-9]|1[0-9]|2[0-9]|3[0-1])[HM]{1}
(AS|BC|BS|CC|CS|CH|CL|CM|DF|DG|GT|GR|HG|JC|MC|MN|MS|NT|NL|OC|PL|QT|QR|SP|SL|SR|TC|TS|TL|VZ|YN|ZS|
E)[B-DF-HJ-NP-TV-Z]{3}[0-9A-Z]{1}[0-9]{1}$
TknRFC ^([A-Z,N,&]{3,4}([0-9]{2})(0[1-9]|1[0-2])(0[1-9]|1[0-9]|2[0-9]|3[0-1])[A-Z\\d]{3})$
TknFecha ^\\d{2}/\\d{2}/\\d{4}$
TknHora ^([01][0-9]|2[0-3]):[0-5][0-9]$
TknFecha-Hora ^\\d{2}/\\d{2}/\\d{4}-([01][0-9]|2[0-3]):[0-5][0-9]$
TknID ^[a-zA-Z_][a-zA-Z0-9_]*$
TknCadena ^^([^\\"\\]*(\\\"\\\"*))"$

```

En esta imagen podemos ver el contenido del archivo de la tabla de tokens en este caso vemos que a la izquierda esta el nombre del token y a la derecha esta la expresión regular del patrón.



```
variables:
variable edad entero
variable nombre texto
codigo-principal:
inicio.
edad = 25
nombre = "Juan"
validar_edad ( mensaje , edad )
fin.
funciones:
funcion saludar ( nombre )
inicio:
imprimir ( "Hola, " + nombre )
devolver
fin:
procedimientos:
procedimiento validar_edad ( mensaje , edad )
inicio:
# Esto es un comentario:
si edad > 18 entonces
imprimir ( "Eres mayor de edad" )
sino
imprimir ( "Eres menor de edad" )
saludar ( nombre )
fin:
CAFA021017HBCRLNA5
CAUC7403314Q0
20/10/2001
10:50
20/10/2002-10:40
```

Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8

En esta imagen podemos ver el contenido del archivo de código que fue proporcionado por el profesor.

TknVariables	TknParentesisCierra
TknVariable	TknSino
TknID	TknImprimir
TknEntero	TknCurp
TknVariable	TknRFC
TknID	TknFecha
TknTexto	TknHora
TknCodigoPrincipal	TknFecha-Hora
TknInicio	Fin del archivo
TknID	PS C:\Users\andre\OneDrive\
TknAsignacion	
TknNumero	
TknID	
TknAsignacion	
TknCadena	
TknID	
TknParentesisAbre	
TknMensaje	
TknComa	
TknID	
TknParentesisCierra	
TknFin	
TknFunciones	
TknID	
TknID	
TknParentesisAbre	
TknID	
TknParentesisCierra	
TknInicio	
TknImprimir	
TknParentesisAbre	
TknCadena	
TknSuma	
TknID	
TknParentesisCierra	
TknDevolver	
TknFin	
TknProcedimientos	
TknProcedimiento	
TknID	
TknParentesisAbre	
TknMensaje	
TknComa	
TknID	
TknParentesisCierra	

Y en esta imagen es lo que nos arroja la consola a la hora de ejecutar el programa, como podemos apreciar en la imagen se mira los nombres de los tokens que detecta el programa en el código.

Conclusión:

Para concluir podemos conocer la importancia del analizador léxico en nuestra área que es la programación ya que este es la primera fase de un compilador.

Para hablar un poco del desarrollo del programa lo primero que hice fue la clase token este fue el mas sencillo de hacer ya que muchas cosas que iban dentro de esta clase fueron encontradas en el Moodle de la clase, después hice la clase nodo que este también fue sencillo de hacer ya que muchas de las cosas que están en el fueron proporcionadas por el profesor, una vez hecha la clase nodo hice la clase lista en la que tuve muchos problemas a la hora de hacer el llenado de la lista ya que no recordaba bien como hacer las listas y como hacer el llenado de esta pero después de volver a investigar sobre esto, también en la clase archivo fue simplemente investigar como se lee un archivo y como se separan las palabras y por último en el main también un poco de investigación y con la ayuda del profesor.

En este trabajo aprendí sobre cómo utilizar las librerías matcher y pattern además de cómo separar las palabras de un archivo y como compararlas con otro archivo. Además de volver a repasar sobre las listas y nodos.

Bibliografía.

2.1. Función del Analizador Léxico. (n.d.). Edu.mx. Retrieved March 12, 2023, from http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro32/21_funcin_del_analizador_lexico.html

De rastreo, L. F., El analizador léxico reconoce, S. de C. Q. R. U. U. de I. en el P. F. E. C. C. un T. R. un C. P. de C. Q., De entrada, o. A. D. el I. de L. C. de E. D. T. M. es N. G. un M. C. Q. N. P. I. el P. de T. E. L. C., Tokens, G., & de estados llamado autómatas finitos., Q. P. S. C. E. M. es P. C. a. P. de un T. E. de M. (n.d.). 2 Análisis léxico (Scanner). Tecnm.Mx. Retrieved March 12, 2023, from <https://hopelchen.tecnm.mx/principal/sylabus/fpdb/recursos/r94255.PDF>

Perfil, V. T. mi. (n.d.). ANALIZADOR LÉXICO. Blogspot.com. Retrieved March 12, 2023, from <https://lenguajesautomatasi.blogspot.com/2015/06/analizador-lexico-y-sus-funciones.html>