



JS

MANUAL TÉCNICO

Programa desarrollado en JAVASCRIPT que permite la lectura de un archivo de texto plano CST que contiene datos acciones a ejecutar simulando un compilador muy básico.

“El manual técnico es el que proporciona todas las pautas de configuración y la lógica con la que se ha desarrollado en el programa, con el objetivo de que cualquier programador entienda la idea propia y su estructura; por lo que se considera necesario ser documentada.”

Elaborado por:

Alvaro Emmanuel Socop Pérez
UNIVERSIDAD DE SAN CARLOS - Facultad de Ingeniería

202000194
28 DE ABRIL DE 2022



ÍNDICE

Introducción.....	2.
Objetivos y alcances del sistema.....	3.
Especificaciones Técnicas.....	4.
Requisitos de Hardware.....	4.
Requisitos de software.....	4.
Sistema operativo.....	4.
Lenguaje de Programación e IDE.....	5.
Tecnologías utilizadas (Lógica del programa).....	5
Funciones utilizadas	7
Flujo del programa.....	9
Descripción del flujo.....	9
Autómata	10
Método del Arbol	11

INTRODUCCIÓN

El presente manual técnico tiene como finalidad describir la estructura y diseño del programa analizador de escritorio de un archivo .EXP, así como dar explicación de los métodos, clases y procesos de cada apartado del programa y la modificación que se le podría dar para cualquier finalidad como el de mejorar el sistema o cambiar algunos atributos propios del analizador. El sistema cuenta con implementación de varias librerías propias de JAVA como Jflex y Jcup, entre otros (Graphviz), como parte del conocimiento adquirido en los laboratorios de Organización de lenguajes de compiladores 1, en base a ello trataré de explicar en que métodos se fueron utilizando y como hacen funcionar el programa en los distintos sistemas operativos.

La implementación de librerías externas a JAVA también fueron una opción para poder pedir y abrir ventanas para elegir los archivos necesarios, también se explicará paso a paso cual es el camino para entender perfectamente la arquitectura del programa. Resulta ser bastante fácil de implementar los ciclos y condiciones, también los métodos de interfaces gráficas puesto que es una función que viene integrada con la paquetería por defecto de JFrame de JAVA y así solo instanciarlos importando las librerías y utilizarlas.

OBJETIVOS Y ALCANCES DEL PROGRAMA

Generales

- ✓ Aplicar conocimientos de programación básica en el lenguaje de programación Java.
- ✓ Aplicar los conocimientos sobre la fase de análisis léxico y sintáctico de un compilador para construcción de una solución de software que permita generar análisis por medio del método del árbol.

Específicos

- ✓ Aprender sobre el uso de los ciclos en Java.
- ✓ Reforzar el concepto del método de Árbol de expresiones regulares en Autómatas Finitos Deterministas (AFD).
- ✓ Reforzar el concepto del método de Thompson de expresiones regulares en Autómatas Finitos No Deterministas (AFND).
- ✓ Identificar y programar el proceso de reconocimiento de lexemas mediante el uso de Autómatas Finitos Deterministas.
- ✓ Desarrollar un analizador de un archivo con extensión .EXP en el lenguaje de programación JAVA para familiarizarse con el lenguaje y los nuevos conceptos aprendidos en el laboratorio.

Dirigido hacia los estudiantes, auxiliares y catedráticos encargados de los Cursos de Lenguajes Formales y de Programación para que verifiquen el método de Thompson y el Método del árbol en mi programa, como forma de entretenimiento y motivación para saber lo que podemos hacer con la programación básica.

ESPECIFICACIÓN TÉCNICA

Requisitos de Hardware

Memoria mínima	512 MB
Memoria recomendada	1 GB
Espacio en disco mínimo	250 MB de espacio libre
Espacio en disco recomendado	500 MB de espacio libre
MVP	Java + JDK instalado

Requisitos de software

• Sistema operativo

Windows

- Windows 10 (8u51 y superiors)
- Tener instalado el programa de Visual Studio Code u otro editor
- RAM: 128 MB
- Espacio en disco: 124 MB
- Procesador: Mínimo Pentium 2 a 266 MHz
- Java y JDK para compilar el programa instalado

Mac OS X

- Tener instalado el programa Visual Studio Code u otro editor
- Explorador de 64 bits
- Se requiere un explorador de 64 bits (Safari, Firefox, por ejemplo) para ejecutar Oracle Java en Mac OS X.

Linux

- Oracle Linux 5.5+1
- Oracle Linux 6.x (32 bits), 6.x (64 bits)2
- Exploradores: Firefox
- Java y JDK para compilar el programa instalado

- **Lenguaje de Programación e IDE**

Para el desarrollo del programa se utilizó el lenguaje de Programación JAVA y el IDE Netbeans y Visual Studio Code.

- **Tecnologías utilizadas**

JAVA es una tecnología que se usa para el desarrollo de aplicaciones que convierten a la Web en un elemento más interesante y útil. Java no es lo mismo que javascript, que se trata de una tecnología sencilla que se usa para crear páginas web y solamente se ejecuta en el explorador.

Dentro de estas funcionalidades básicas de JAVA encontramos: el uso de colecciones, acceso a ficheros con IO, con la característica de que es un lenguaje de programación multiparadigma, librerías para el desarrollo de aplicaciones de escritorio o web como JFlex y Jcup, capacidades para realizar analisis sintáctico y Léxico...



LÓGICA DEL PROGRAMA

Explicación de la lógica del programa:

El programa comienza con una interfaz general el cual luego de cargar el archivo lfp en la cinta de opciones se llama al método Analizefile. El cual por medio de un automata implementado y utilizando validaciones se lee el archivo de texto que se ha cargado al sistema.

Posteriormente se procede a recorrer cada uno de los caracteres del archivo para validar el lenguaje y que sea correcto (análisis léxico), luego se recorren varios métodos para validar las posiciones que tiene el archivo y así confirmar que el archivo esta escrito correctamente, sin ningún error.

Dentro del programa se tiene la opción de volver a compilar el programa esto con el fin de demostrar que es posible recompilar en tiempo real y editando el texto en la interfaz gráfica todas las instrucciones que se deseen ingresar en el archivo de texto plano.



Funciones utilizadas (Métodos)

- **Arbol**

Se crea un BST con los nodos recogidos para crear el arbol

- **Transiciones y siguientes**

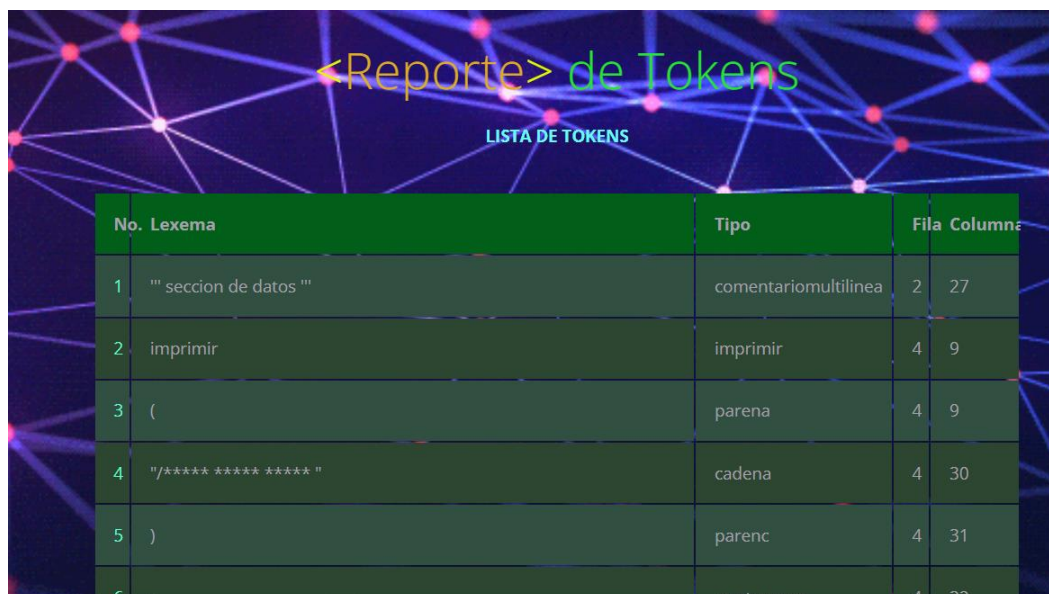
Se crea a partir del arbol una tabla de siguientes y una de transiciones

- **Thompson**

Se recorre el arbol y usando recursividad se va creando por medio de subgrafos el metodo de Thompson

- **Reportes**

Se cuenta con una pestaña de reportes el cual muestra los Tokens y Errores que se guardaron en el análisis léxico del archivo exp, detallando que token, lexema fila y columna se encuentra cada error o Token



The screenshot shows a web application interface with a dark blue background featuring a network of glowing nodes and lines. At the top, the title "<Reporte> de Tokens" is displayed in a stylized font, with "<Reporte>" in orange and "de Tokens" in green. Below the title, the text "LISTA DE TOKENS" is visible in white. A table with a green header and dark green body is shown, containing six rows of token data. The table has four columns: "No.", "Lexema", "Tipo", and "Fila Columna".

No.	Lexema	Tipo	Fila	Columna
1	" seccion de datos "	comentariomultilinea	2	27
2	imprimir	imprimir	4	9
3	(parena	4	9
4	"/***** *****/	cadena	4	30
5)	parenc	4	31
6	.	puntocoma	4	32

Flujo del programa y Descripción del flujo

Main1(): Se carga la interfaz gráfica

Ventana Principal (): se muestran todos los componentes en ventana

MyLexer (): Se analiza las líneas cargadas del archivo exp.

analizar(allines) Analiza sintácticamente todas las líneas del archivo lfp

Sintactic_analyzer: Recibe como parametro el índice de token a iniciar a analizar sintácticamente.

Transitions and Nexts (methods) Ordenan los datos y los agrega a tablas las cuales se muestran en la tabla generada como html del programa.

CargarListas: Se recorre nuevamente todas las funciones que se utilizaron en el análisis sintáctico, pero esta vez realizando todo el proceso de compilación para guardarlo en la cadena de Salida que se mostrara en la CONSOLA simulada.

Listado de Tokens

TOKEN	PATRON	TIPO
variable	Cualquier carácter inicia en letra o guion bajo seguido de letras o números (reservada)	léxico
letra	Una sola letra mayuscula o minuscula	lexico
entero	Secuencia de dígitos	léxico
llavea	Carácter llave que abre {	léxico
llavec	Carácter llave que cierra }	léxico
puntocomma	Carácter Punto y coma ;	léxico
coma	Coma (,)	léxico
cadena	Inicia en “ o ‘ y finaliza en “ o ‘	léxico
error	Cualquier carácter que no esté en el lenguaje	léxico
comentarios	Inicia en <! y finaliza en !> o solo inicia en //	
mayorque	Carácter >	léxico
ENTERO	Palabra Int(reservada)	léxico
DOUBLE	Palabra double(reservada)	léxico
BOOLEANO	Palabra boolean (reservada)	léxico
CARACTER	Palabra char (reservada)	léxico
VOID	Palabra void (reservada)	léxico
STRING	Palabra string (reservada)	léxico

TRUE	Palabra true (reservada)	léxico
STRING	Palabra false (reservada)	léxico
NEW	Palabra new (reservada)	léxico
INC	Carácter ++	léxico
DEC	Carácter --	léxico
MAS	Carácter +	léxico
MENOS	Carácter -	léxico
MULTIPLICACION	Carácter *	léxico
DIVISION	Carácter /	léxico
POTENCIA	Carácter ^	léxico
MODULO	Carácter %	léxico
IGUALA	Carácter ==	léxico
DIFERENTE	Carácter !=	léxico
MENORIGUALQ	Carácter >=	léxico
MENORQUE	Carácter >	léxico
MAYORIGUALQ	Carácter <=	léxico
MAYORQUE	Carácter <	léxico
IGUAL	Carácter =	léxico
OR	Carácter	léxico

AND	Carácter	léxico
NOT	Caracter j	
MAYORIGUALQ	Carácter >=	léxico
IF	Palabra if (reservada)	léxico
ELSE	Palabra else (reservada)	léxico
SWITCH	Palabra switch (reservada)	léxico
CASE	Palabra case (reservada)	léxico
BREAK	Palabra break (reservada)	léxico
DEFAULT	Palabra default (reservada)	léxico
WHILE	Palabra while (reservada)	léxico
DO	Palabra do (reservada)	léxico
FOR	Palabra for (reservada)	léxico
CONTINUE	Palabra continue (reservada)	léxico
RETURN	Palabra return (reservada)	léxico

Expresiones Regulares

Para la lectura del archivo se utilizaron varias expresiones regulares entre las que se pueden mencionar:

- 1) **numero**=[0-9]+
- 2) **variables**=[a-z|A-Z]+[a-z|A-Z|0-9|"_"]+
- 3) **letrasmin**=[a-z] (**letras minusculas**)
- 4) **comentlinea** = ("/*.*\r\n)|("/*.*\n)|("/*.*\r)
- 5) **comentarios**="<![!]*([^!>]|([!]">"|!"["^>]))*!"!>"
- 6) **cadena**s= ["\[^\"]*["]
- 7) **salto linea**="\n"
- 8) **comillas simple**reg="\\\" (**comillas simples**)
- 9) **comillas doble**reg="\\\"" (**comillas dobles**)
- 10) **especial** = ("\\n|\"\\\"|\"'|\"\\\\\"|\"\\\\\\\\\"|\"\\\\\\\\\\\\\"|\"\\\\\\\\\\\\\\\\\"") (**caracteres especiales**)
- 11) **VDOUBLE**= [0-9]+(."[0-9]+)\b
- 12) **VENTERO**= [0-9]+\b**numeros**)
- 13) **VCARACTER** =
(\'((\\[\\\"\\\'\\`\\~\\!\\@\\#\\\$\\%\\&*\\(\\)\\+\\,\\-\\.\\/\\:;\\<\\>\\=\\?\\[\\]\\^_\\`\\{\\|\\}~) | ([^\\n\\\"\\\'\\`\\~\\!\\@\\#\\\$\\%\\&*\\(\\)\\+\\,\\-\\.\\/\\:;\\<\\>\\=\\?\\[\\]\\^_\\`\\{\\|\\}~]))\'

Lo demás fue utilizado con el análisis sintáctico

/*-----5.10 Precedencia de Operaciones-----*/

Asociación de operadores y precedencia

La precedencia de operadores nos indica la importancia en que una operación debe realizarse por encima del resto. A continuación, se define la misma.

'OR', 'INTERROGA', 'tipo'

'AND'

'NOT'

'IGUALA', 'DIFERENTE', 'MENORQUE', 'MENORIGUALQ', 'MAYORQUE', 'MAYORIGUALQ'

'INC', 'MAS', 'DEC', 'MENOS'

'MULTIPLICACION', 'DIVISION', 'POTENCIA', 'MODULO', 'VARIABLE'

UMENOS

Gramática Independiente del Contexto

GRAMATICA COMENTARIOS

COMENTARIOS1 = menor que signo admira cadena signo admira mayor que <! cadena- !>

COMENTARIOS2 = slash slash cadena // cadena

Gramática independientes de contexto

/* ===== Definición de la gramática ===== */

ini : instrucciones EOF

;

instrucciones

: instrucciones instruccion

| instruccion

;

/* ===== INSTRUCCIONES ===== */

instruccion

: declaracion

| funciones

| metodos

| call

| instruccioneswitch

| instruccionif

| instruccionwhile

| instruccionfor

| instrucciondowhile

| instruccionprint

```

| instruccionprintln

| VARIABLE INC PTCOMA

| VARIABLE DEC PTCOMA

| CONTINUE PTCOMA

| BREAK PTCOMA

| returns PTCOMA

| RUN call

/* | incredecre { } 5.14 Incremento y Decremento */

;

```

/*-----5.17 Sentencias cíclicas-----*/

/*----- 5.19 Funciones -----*/

funciones

```

: VARIABLE PARIZQ parametros PARDER DOSPUNTOS tipo LLAIZQ instrucciones LLADER

| VARIABLE PARIZQ PARDER DOSPUNTOS tipo LLAIZQ instrucciones LLADER

;

```

/*----- 5.20 Metodos -----*/

metodos

```

: VARIABLE PARIZQ parametros PARDER DOSPUNTOS VOID LLAIZQ instrucciones LLADER

| VARIABLE PARIZQ parametros PARDER LLAIZQ instrucciones LLADER

```


| VARIABLE PARIZQ PARDER DOSPUNTOS VOID LLAIZQ **instrucciones** LLADER

| VARIABLE PARIZQ PARDER LLAIZQ **instrucciones** LLADER

;

/*----- 5.21 Llamadas -----*/

call

:**llamadas** PTCOMA

;

llamadas

:VARIABLE PARIZQ **paramllamada** PARDER

|VARIABLE PARIZQ PARDER

;

paramllamada

:**paramllamada** COMA **expresion**

|**expresion**

;

parametros

:**parametros** COMA **tipo** VARIABLE

|**tipo** VARIABLE

;

/*----- IF -----*/

instruccionif

:IF PARIZQ **expresion** PARDER LLAIZQ **instrucciones** LLADER

|IF PARIZQ **expresion** PARDER LLAIZQ **instrucciones** LLADER ELSE LLAIZQ **instrucciones** LLADER

|IF PARIZQ **expresion** PARDER LLAIZQ **instrucciones** LLADER ELSE **instruccionif**

|IF PARIZQ **expresion** PARDER LLAIZQ LLADER

;

/* ----- SWITCH ----- */

instruccionswitch

:SWITCH PARIZQ **expresion** PARDER LLAIZQ **instruccioncaselist instrucciondefault** LLADER

|SWITCH PARIZQ **expresion** PARDER LLAIZQ **instruccioncaselist** LLADER

|SWITCH PARIZQ **expresion** PARDER LLAIZQ **instrucciondefault** LLADER

;

instruccioncaselist

:**instruccioncaselist** CASE **expresion** DOSPUNTOS **instrucciones**

|CASE **expresion** DOSPUNTOS **instrucciones**

;

/* ----- DEFAULT ----- */

instrucciondefault

:DEFAULT DOSPUNTOS **instrucciones** { }

;

/*-----5.17.1. While -----*/

instruccionwhile

```
:WHILE PARIZQ expresion PARDER LLAIZQ instrucciones LLADER  
;
```

/*-----5.17.3. Do-While-----*/

instrucciondowhile

```
:DO LLAIZQ instrucciones LLADER WHILE PARIZQ expresion PARDER PTCOMA  
;
```

/*-----5.17.2. For-----*/

instruccionfor

```
:FOR PARIZQ fordeclarar PTCOMA expresion PTCOMA actualizacion PARDER LLAIZQ  
instrucciones LLADER  
;
```

fordeclarar

```
: tipo VARIABLE IGUAL expresion  
| VARIABLE IGUAL expresion  
;
```

actualizacion

```
: VARIABLE INC  
| VARIABLE DEC  
| VARIABLE IGUAL expresion  
;
```

/*----- 5.22 Función Print -----
*/

instruccionprint

:PRINT PARIZQ **expresion** PARDER PTCOMA
;

/*----- 5.23 Función Println -----
---*/

instruccionprintln

:PRINTLN PARIZQ **expresion** PARDER PTCOMA
;

/*----- RUN -----*/

instruccionrun

: RUN VARIABLE PARA PARC
| RUN VARIABLE PARA **listavalores** PARC
;

/*----- RETURN -----*/

returns

: RETURN **expresion**
| RETURN
;

/*----- DECLARACION ASIGNACION VARIABLES -----
----- */

declaracion

```

: tipo VARIABLE PTCOMA

| tipo VARIABLE IGUAL expresion PTCOMA

| tipo notacioncomas PTCOMA

| tipo notacioncomas IGUAL expresion PTCOMA

| tipo VARIABLE CORIZQ CORDER IGUAL NEW tipo CORIZQ expresion CORDER PTCOMA

| tipo VARIABLE CORIZQ CORDER IGUAL NEW tipo CORIZQ expresion CORDER CORIZQ
expresion CORDER PTCOMA

| tipo VARIABLE CORIZQ CORDER IGUAL CORIZQ listavalores CORDER PTCOMA

| tipo VARIABLE IGUAL VARIABLE CORIZQ expresion CORDER PTCOMA

| tipo VARIABLE IGUAL VARIABLE CORIZQ expresion CORDER CORIZQ expresion CORDER
PTCOMA

| notacioncomas IGUAL expresion PTCOMA

| VARIABLE IGUAL expresion PTCOMA

/*5.15.1.3 Modificación de Vectores*/

| VARIABLE CORIZQ expresion CORDER IGUAL expresion PTCOMA

| VARIABLE CORIZQ expresion CORDER CORIZQ expresion CORDER IGUAL expresion
PTCOMA

;

```

notacioncomas

```

: notacioncomas COMA VARIABLE

| VARIABLE

;

```

/* ----- EXPRESIONES ----- */

expresion

: MENOS expresion

| NOT expresion

| expresion MAS expresion

| expresion MENOS expresion

| expresion MULTIPLICACION expresion

| expresion DIVISION expresion

| expresion MODULO expresion

| expresion POTENCIA expresion

| expresion MENORIGUALQ expresion

| expresion MENORQUE expresion

| expresion MAYORIGUALQ expresion

| expresion MAYORQUE expresion

| expresion IGUALA expresion

| expresion DIFERENTE expresion

| expresion OR expresion

| expresion AND expresion

| PARA tipo PARC expresion

| VARIABLE

| VENTERO

| VDOUBLE
 | CADENA
 | VCHARACTER
 | TRUE
 | FALSE
 | PARIZQ **expresion** PARDER
 | VARIABLE CORIZQ **expresion** CORDER
 | VARIABLE CORIZQ **expresion** CORDER CORIZQ **expresion** CORDER
 | **expresion** INTERROGA **expresion** DOSPUNTOS **expresion**
 | TOLOWER PARIZQ **expresion** PARDER
 | TOUPPER PARIZQ **expresion** PARDER
 | ROUND PARIZQ DOUBLE PARDER
 | TYPEOF PARIZQ **expresion** PARDER
 | TOSTRING PARIZQ **expresion** PARDER
 | LENGTH PARIZQ **expresion** PARDER
 | TYPEOF PARIZQ **expresion** PARDER
 | TOSTRING PARIZQ **expresion** PARDER
 | TOCHARARRAY PARIZQ **expresion** PARDER
 | **llamadas**

;

listavalores

: **listavalores** COMA **expresion**

| **expresion**

;

/* ----- TIPOS ----- */

tipo

:ENTERO

|DOUBLE

|BOOLEANO

|CARACTER

|STRING

|VOID

;