Manual Técnico

“Proyecto 2 [LFP 2]”

Facultad de Ingeniería

Septiembre, 2021

Proyecto 2 [LFP 2]

Autor:

Sergie Daniel Arizandieta Yol

202000119

Facultad de ingeniería

Lenguajes Formales y de programación

Universidad de San Carlos de Guatemala

Guatemala, octubre 2021

OBJETIVOS

1. Objeto del documento

El documento tiene como finalidad proporcionar información de manera técnica para todo aquel interesado en la edición del código fuente del software con la cual puede conocer el manejo adecuado y correcto que la aplicación necesita para el funcionamiento de dicho programa.

2. Objetivos

- Otorgar al usuario una explicación simple y concisa de entender todas las características y el diseño de software de forma técnica del software que posee de manera lógica.

- Entregar al lector los fundamentos teóricos del software presentado buscando que sea apoyo para la compresión del código base a los fundamentos a presentar en el documento.

- Que todo usuario que utilice o busque optimización y edición del software sea capaz de cumplirlo entendiendo las bases con la que este funciona.

INTRODUCCIÓN

El manual técnico tiene como finalidad dar a conocer al lector e interesado en el mismo que pueda requerir modificaciones futuras el desarrollo del software “Analizador y reportador de datos” indicando de una forma técnica características y requerimientos de este.

El software tiene como objetivo realizar la lectura de los datos mediante un analizador léxico, procesado y atravesando por un analizador sintáctico de un archivo especifico en el cual provee todos los requerimientos al software para que este pueda manipular, crear reportes y creación de archivos de salida solicitados mediante este, así mismo con la posibilidad de exportación de reportes de los datos mediante métodos efectivos y rápidos.

DESCRIPCIÓN DE LA SOLUCIÓN

Es una aplicación enfocada a la lectura y salida de archivos a través de una Interfaz de Usuario (UI), donde dicho software se divide en tres partes, siendo la principal la lectura del archivo de entrada desplegándolo en la interfaz para que usuario pueda interactuar con dicha información, así mismo luego se procede a la lectura de dicha data siento consumidas por un analizador léxico y consecuentemente un analizador sintáctico, siendo capaz de capturar errores en el documento y siendo capaz de consultarlos mediante reportes, ya siendo esta sección concluida se procede a los archivos de salida.

Según los requerimientos de los archivos de entrada el software procesará los datos de maneras específicas y siendo capaces de consultar los archivos de salida generados en este caso las respuesta de las instrucciones indicadas por el archivo de salida, así mismo como la exportación de reporte de la lectura tanto de tokens como de errores del archivo de entrada el cual debe cumplir con usa estructura específica para el funcionamiento de este, el cual será presentado a continuación, además del árbol de derivación de la gramática.

Requerimientos archivo de entrada

Para el funcionamiento de este los archivos de entrada debe cumplir con la estructura presentada en las siguientes figuras con extensión “flp”.

Se debe contener las Claves consecuente mente los Registros y para concluir las instrucciones a realizar, para las Claves la estructura es la siguiente:

Ejemplar Claves:

```
Claves = [
    "clave_1", "clave_2", "clave_3", "clave_4"
]
```

Consecuentemente se deben ingresar los registros para las claves de la siguiente manera: donde los valores pueden ser de las siguientes formas “valor_ingresado” o un digito: 1,2…100 con la siguiente estructura:

Ejemplar Registros

```
Registros = [
    {valor1, valor2, valor3, valor4}
    {valor1, valor2, valor3, valor4}
    {valor1, valor2, valor3, valor4}
    {valor1, valor2, valor3, valor4}
]
```

Funciones
Siempre y cuando de hayan ingresado correctamente las claves y registros se podrán ejecutar las siguientes funcionalidades:

Comentarios:

- Comentarios de una línea: Se representan con un numeral y finalizan con un salto de línea.

```
# Comentarios
```

- Comentarios multilínea: Inicia con tres comillas simples y finaliza con tres comillas simples.

```
'''
comentario multilinea
'''
```

Instrucciones de Reportería:

● imprimir(cadena): Imprime por consola el valor dado por la cadena en una misma linea.

```
imprimir("Reporte de ");
imprimir("Abarroteria");
>>> Reporte de Abarroteria
```

● imprimirln(cadena): Imprime por consola el valor dado por la cadena con un salto de línea

```
imprimirln("Reporte de ");
imprimirln("Abarroteria");
>>> Reporte de Abarroteria
>>> Abarroteria
```

● conteo(): Imprime por consola la cantidad de registros en el arreglo de registros.

```
conteo();
>>> 46
```

● promedio(“campo”): Imprime por consola el promedio del campo dado.

```
promedio("stock");
>>> 6.25
```

● contarsi(“campo”, valor): Imprime por consola la cantidad de registros en la que el campo dado sea igual al valor dado.

```
contarsi("stock", 0);
>>> 0
```

● datos(): Imprime por consola los registros leídos

```
datos();
>>> codigo  producto  precio_compra precio_venta stock
>>> 1       Barbacoa  10.50         20.00        6
>>> 2       Salsa     13.00         16.00        7
>>> 3       Mayonesa  15.00         18.00        8
>>> 4       Mostaza   14.00         16.00        4
```

● sumar(“campo”): Suma todos losvalores del campo dado.

● max(“campo”): Encuentra el valor máximo del campo dado.

```
max("precio_venta");
>>> 20.00
```

● min(“campo”): Encuentra el valor mínimo del campo dado.

```
min("precio_compra");
>>> 10.50
```

● exportarReporte(“titulo”): Genera un archivo html con una tabla en donde se encuentren los registros leídos y con el título como parámetro.

titulo				
codigos	producto	precio_compra	precio_venta	stock
1	Barbacoa	10.50	20.00	6
2	Salsa	12.00	16.00	7
3	Mayonesa	15.00	18.00	4
4	Mostaza	14.00	16.00	4

Ejemplar de una entrada con todas las funciones

```
Claves = [

        "codigos" ,"producto", "precio_compra","precio_venta","stock"
]

Registros = [
        {1,"Barbacoa",10.50,20.00,6}
        {2,"Salsa",12.00,16.00,7}
        {3,"Mayonesa",15.00,18.00,4}
        {4,"Mostaza",14.00,16.00,4}

]
```

```
imprimir("Plantilla actual de: ");
imprimirln("FC Bayern Munchen");

imprimirln("conteos");
conteo();

imprimirln("promedio");
promedio("edad");

imprimirln("contar si");
contarsi("nacionalidad", "Alemán");

imprimirln("datos");
datos();

imprimirln("sumar edades");
sumar("edad");

imprimirln("mayor edad");
max("edad");

imprimirln("menor edad");
min("edad");

exportarReporte("titulo");
```

Respuesta:

```
Plantilla actual de:
FC Bayern Munchen
conteos
20
promedio
No se encontro registro
contar si
No se encontro registro
datos
  codigos  producto precio_compra precio_venta stock
0       1  Barbacoa         10.50        20.00     6
1       2     Salsa         12.00        16.00     7
2       3  Mayonesa         15.00        18.00     4
3       4   Mostaza         14.00        16.00     4

sumar edades
No se encontro registro
mayor edad
No se encontro registro
menor edad
No se encontro registro
Se genero la exportacion del Reporte
```

titulo				
codigos	producto	precio_compra	precio_venta	stock
1	Barbacoa	10.50	20.00	6
2	Salsa	12.00	16.00	7
3	Mayonesa	15.00	18.00	4
4	Mostaza	14.00	16.00	4

Tabla de Tokens

Se implemento una lista de Tokens para la lectura del archivo de entrada, Tabla 1, la cual está regida según el alfabeto:

- S = {‘=’ , ‘;’, ‘ , ’ , ‘{‘ , ‘ [‘ , ‘]’ , ’ }’ }
- L ={a-zA-ZñÑ}
- D = {0-9}

Token	Patrón	Lexema de Ejemplo
ID	Id = L(L|D|'_')*	Texto
Registros	(' " ' (^")* ' " ')	“Titulo”
Símbolo	S	;
Numero	D+(“.”D+)?	90.00
Comentario	(' # ' (^\n)* ' \n ')	#Hola mundo
ComentarioMultilinea	(“ ‴ “ (^‴)*“ ‴ “)	‴ Hola A todos ‴

Tabla 1.

Analizador léxico

Es un módulo destinado a leer caracteres del archivo de entrada, donde se encuentra la cadena a analizar que correspondan a símbolos del lenguaje y retornar los tokens correspondientes y sus atributos, siempre apegado a una expresión regular o un Autómata Finito Determinista.

Proceso método del árbol

Expresión regular

Es una cadena de caracteres que es utilizada para describir o encontrar patrones del archivo de entrada.

La expresión regular (ER) utilizada fue:

Proceso expresión regular

Definir alfabeto a utilizar:

- Símbolos = {= , ; , ‘,’, { , [,] , } }
- L ={a-zA-ZñÑ}
- D = {0-9}

Definir Expresión Regular:

- ER = (ID | Símbolos | Registros | Numero | Comentario | ComentarioMultilinea) $

Árbol binario

De esta misma se procedió a generar un árbol binario el cual es una estructura de datos de forma binaria es decir solo puede tener 2 hijos por cada nodo que este contenga, teniendo él cuenta la expresión regular anterior la Figura 1, es la representación en árbol binario de la expresión regular anteriormente definida,

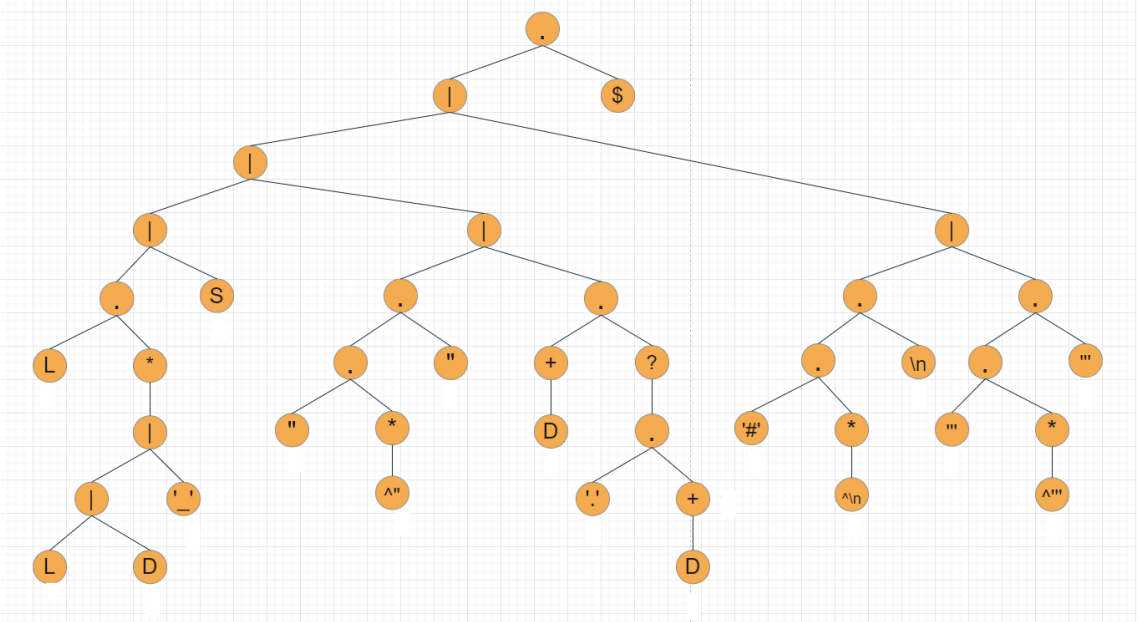
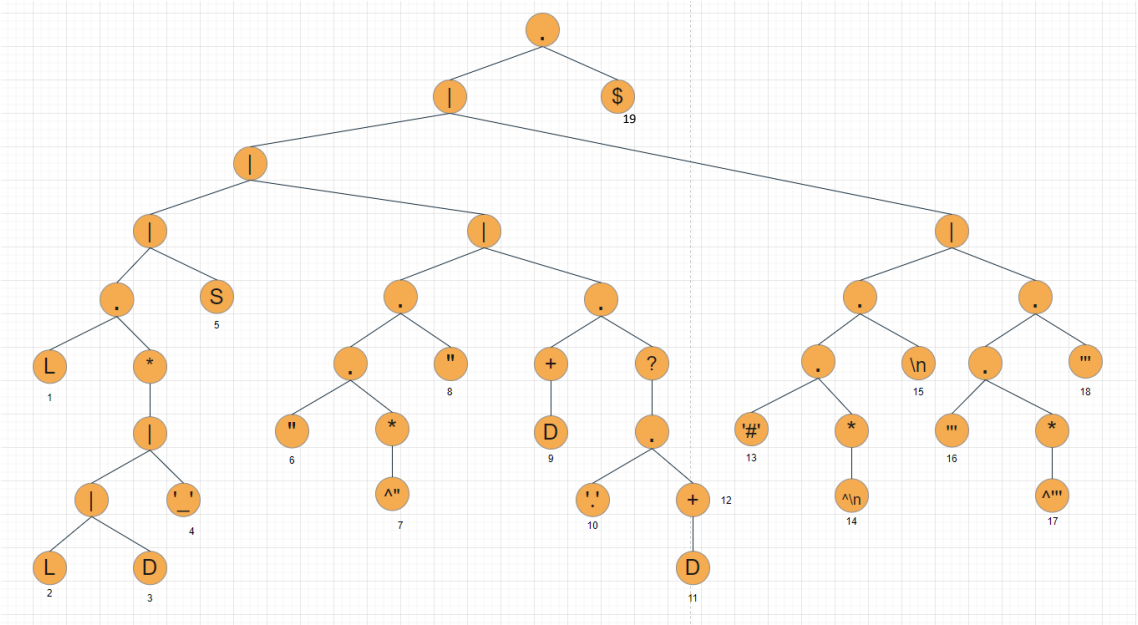


Figura 1

Enumerar los hijos del nivel más bajo

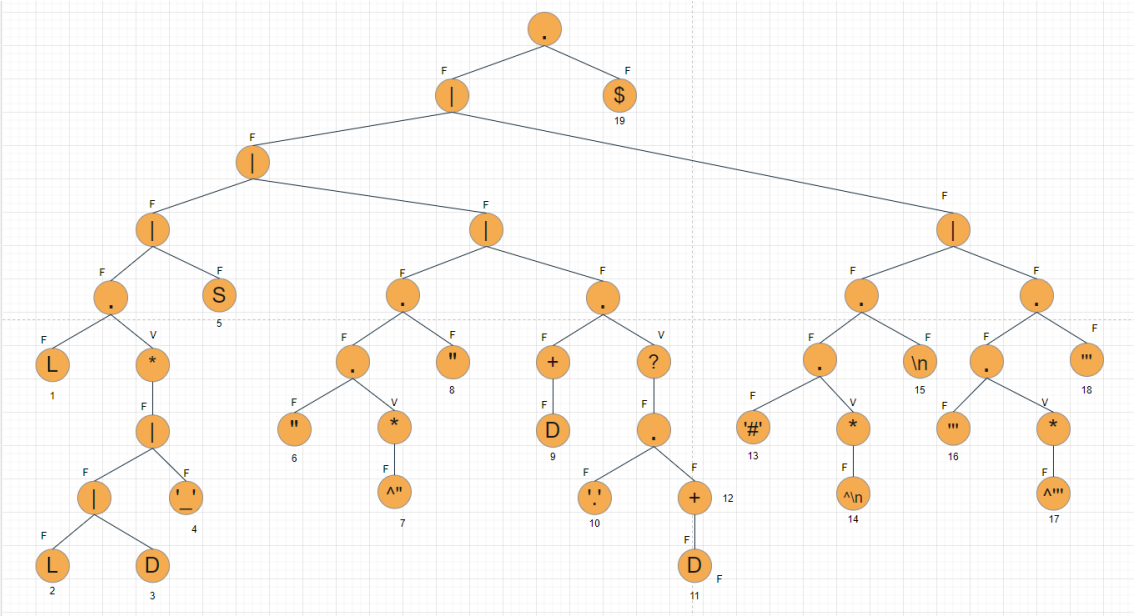


Calcular Anulables

Se procedió a definir los nodos anulables del árbol según la tabla siguiente:

Nodo	Anulable
C1 (hoja)	**F**
C1 | C2	If isNullable(C1) || isNullable(C2) **V** else **F**
C1 C2	If isNullable(C1) && isNullable(C2) **V** else **F**
C1*	**V**
C1+	If isNullable(C1) **V** else **F**
C1?	**V**

Fuente: Laboratorios Lenguajes Formales y de programación Sección B-

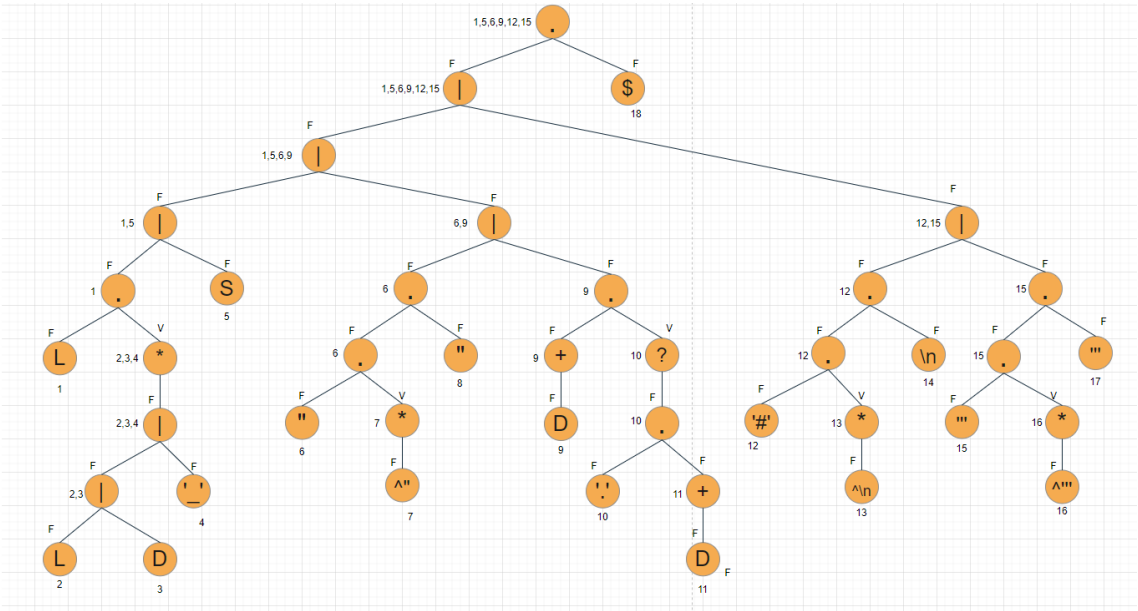


Calcular Primeros

Se procedió a definir los nodos con sus primeros del árbol según la tabla siguiente:

Nodo	Primeros ó F(x)
C1 (hoja)	**C1**
C1 | C2	**F(C1) U F(C2)**
C1 C2	If isNullable(C1) **F(C1) U F(C2)** else **F(C1)**
C1*	**F(C1)**
C1+	**F(C1)**
C1?	**F(C1)**

Fuente: Laboratorios Lenguajes Formales y de programación Sección B-

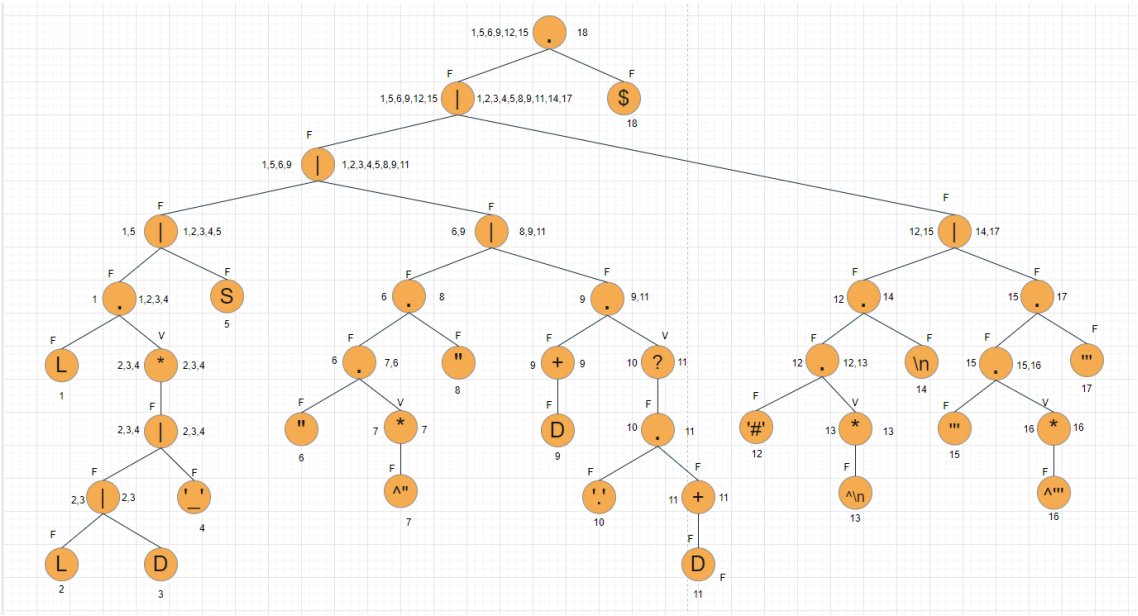


Calcular Últimos

Se procedió a definir los nodos con sus últimos del árbol según la tabla siguiente:

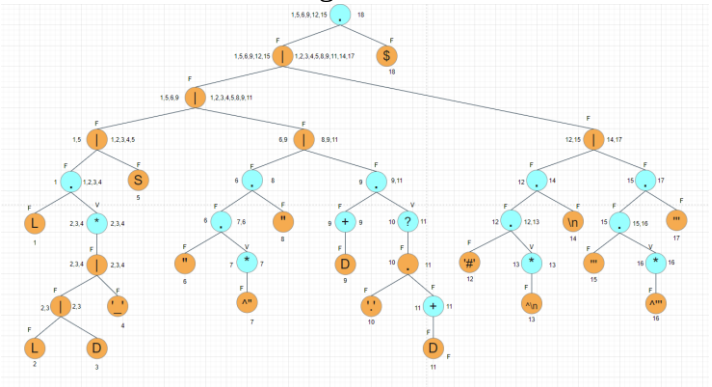
Nodo	Últimos ó L(x)
C1 (hoja)	**C1**
C1 | C2	**L(C1) U L(C2)**
C1 C2	If isNullable(C2) **L(C1) U L(C2)** else **L(C2)**
C1*	**L(C1)**
C1+	**L(C1)**
C1?	**L(C1)**

Fuente: Laboratorios Lenguajes Formales y de programación Sección B-



Calcular Siguientes

Se procedió a definir los nodos con sus siguientes del árbol en los nodos *, + y • :



Valor	Hoja	Siguiente
L	1	2,3,4,18
L	2	2,3,4,18
D	3	2,3,4,18
_	4	2,3,4,18
S	5	18
"	6	7,8
^"	7	7,8
"	8	18
D	9	9,10,18
.	10	11
D	11	11,18
#	12	13,14
^\n	13	13,14
\n	14	18
'''	15	16,17
^'''	16	16,17
'''	17	18
$	18	

Crear Tabla de Transiciones

Se procedió a establecer la tabla de transmisiones de estados para el autómata:

Aceptacion	Estado	Valores	Siguientes
	S0	L,S,",D,#,'''	L(2,3,4,18) = **S1**
		1,5,6,9,12,15	S(18) = **S2**
			"(7,8) = **S3**
			D(9,10,18) = **S4**
			#(13,14) = **S5**
			"'(16,17) = **S6**
aceptacion	S1	L,D,_,$	L(2,3,4,18) =S1
		2,3,4,18	D(2,3,4,18) = S1
			_(2,3,4,18)=S1
			${}
acpetacion	S2	$	${}
		18	
	S3	^","	^"(7,8) = S3
		7,8	"(18) = S2
aceptacion	S4	D,.,$	D(19,10,18) = S4
		9,10,18	.(11) = **S7**
			${}
	S5	^\n,\n	^\n(13,14) = S5
		13,14	\n(18) = S2
	S6	^'','''	^''(16,17) = S6
		16,17	"'(18) = S2
	S7	D	D(11,18) = **S8**
		11	
acptacion	S8	D,$	D(11,18) = S8
		11,18	${}

Tabla de Transiciones para el autómata

Se procedió a establecer la tabla de transmisiones para el autómata:

Aceptacion	Estado (S)	L	D	_	S	"	^"	.	#	^\n	\n	'''	^'''	S
	S0	S1	S4		S2	S3			S5			S6		
$	S1	S1	S1	S1										
$	S2													
	S3					S2	S3							
$	S4		S4					S7						
	S5									S5	S2			
	S6											S2	S6	
	S7		S8											
$	S8		S8											

Autómata Finito Determinista (AFD)

Autómata

Estudio matemático para una máquina de estado finito.

Dada una entrada de símbolos se mueve hasta que es completamente consumida a través de un conjunto de estados según una función de transición.

Autómata Finito

- Conjunto de estados bien definidos.
- Estado inicial
- Estados de aceptación
- Alfabeto
- Transiciones

La cual puede representar de la misma manera a una expresión regular, según la expresión anteriormente propuesta del analizador léxico el Autómata Finito Determinista correspondiente y tomando de guía la **Tabla de Transiciones para el autómata** es el siguiente, ver Figura 2

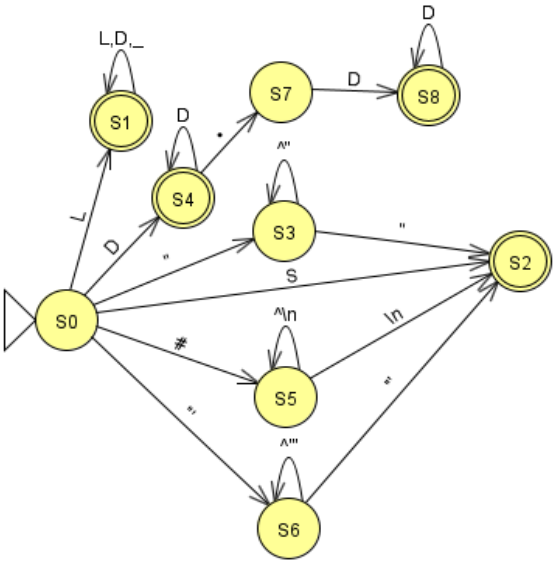


Figura 2

Gramática independiente del contexto (tipo 2)

Gramática

Estructura lógico-matemática con un conjunto de reglas de formación que definen las cadenas de caracteres admisibles en un determinado lenguaje.

Gramáticas tipo 2

Son las que generan los lenguajes libres o independientes del contexto. Los lenguajes libres del contexto son aquellos que pueden ser reconocidos por un autómata de pila determinístico o no determinístico.

Poseen cuatro componentes:

- Conjunto de símbolos terminales o alfabeto. (Tokens)
- Conjunto de no terminales o “variables sintácticas” que representan un conjunto de cadenas o terminales.
- Conjunto de producciones donde se tiene en lado izquierdo un no terminal llamado encabezado, una flecha y en la parte derecha una secuencia de terminales y no terminales llamado cuerpo.

  E -> E + E

- Designación del símbolo o producción inicial

Redefiniendo los tokens y expresiones regulares

Proceso expresion regular		
L ={a-zA-ZñÑ}		
D = {0-9}		
ID = L(L| D| '_')*		
RegistrosTxt = (' " ' (^")* ' " ')		
Numero = D+(“.” D+)?		
Simbolos = {= , ; , { , [,] , } }		
Simbolos extendido		
igual	=	
PtoComa	;	
LA	{	
LC	}	
CA	[	
CC	]	
PA	(	
PC	)	
Coma	,	
Tokes palabras reservadas		
Registro		
Claves		
imprimir		
imprimirln		
conteo		
promedio		
contarsi		
datos		
max		
min		
exportarReporte		

Componentes de la gramática

	Tokens	No teminales	Inicio
Terminales	ID	Inicio	<inicio>
	Registros		
	Numero		
	igual		
	PtoComa		
	LA		
	LC		
	CA		
	CC		

Gramática

<Inicio>	-->	<Claves> <Registros> <Lins>
<Claves>	-->	Claves igual CA <AsiganacionC> CC
<AsiganacionC>	-->	RegistrosTxt <AC'>
<AC'>	-->	(Coma RegistrosTxt <AC'>)|e
<Registros>	-->	Registro igual CA <AsiganacionR> CC
<AsiganacionR>	-->	LA <dataR> LC <AR'>
<AR'>	-->	(LA <dataR> LC <AC'>)|e
<dataR>	-->	<valor> <dR'>
<dR'>	-->	(Coma <valor> <dR'>)|e
<valor>	-->	RegistrosTxt | Numero
<LIns>	-->	<Ins> <Lins'>
<Lins'>	-->	(<Ins> <Lins'>) | e
<Ins>	-->	<imprimir>|<imprimirln>|<conteo>|<promedio>|<contarsi>|<datos>|<max>|<min>|<exportarReporte>|e
<imprimir>	-->	imprimir PA RegistrosTxt PC PtoComa
<imprimirln>	-->	imprimirln PA RegistrosTxt PC PtoComa
<conteo>	-->	conteo PA PC PtoComa
<promedio>	-->	promedio PA RegistrosTxt PC PtoComa
<contarsi>	-->	contarsi PA RegistrosTxt Coma <elemento> PC PtoComa
<elemento>	-->	RegistrosTxt | Numero
<datos>	-->	datos PA PC PtoComa
<max>	-->	max PA RegistrosTxt PC PtoComa
<min>	-->	mins PA RegistrosTxt PC PtoComa
<exportarReporte>	-->	exportarReporte PA ID PC PtoComa

Requerimientos mínimos

Sistema

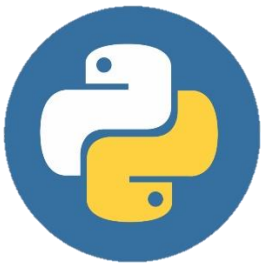
- 2 GB de RAM
- Windows vista
- Arquitectura 32 bits

Editor de Texto – Visual Studio Code

El editor de texto con el que se desarrolló la práctica “Proyecto 1 [LFP 1]” fue Visual Estudio Code V1.59, debido a que el desarrollo y conectividad que puede tener con extensiones y tecnologías como Github es más ventajoso en muchos sentidos por lo cual hace que la fase de programación sea más sencilla y eficaz.

Python 3

El lenguaje con el que se desarrolló “Proyecto 1 [LFP 1]” fue en Python 3.9.6, debido que los lineamientos de la practica así fue establecido.



HTML Y CSS

Los reportes de dicha práctica fueron desarrollados desde Python en HTML5 y CCS3, basado en lineamientos de practica con CSS adicional para una mejor vista al usuario.

Librerías Utilizadas

- **Tkinter:** Librería utilizada para crear interdaces graficas en Python.
- **PIL:** Librería utilizada para brindar soporte para abrir, manipular y guardar muchos formatos de archivos de imagen en Python.
- **Html2image:** Librería es una librería de Python que actúa como envoltorio del modo sin cabeza de los navegadores web existentes para generar imágenes a partir de URL o archivos HTML con CSS.

Módulos

- **Copy:** Incluye dos funciones, copy() y deepcopy(), para duplicar objetos existentes.

Sistema Operativo

El sistema operativo en el que se llevó a cabo el proyecto fue: