Universidad de san Carlos de Guatemala
Lenguajes formales y de programación
Sección b-
Catedrática: inga. Zulma Aguirre
Auxiliar Académico: Cesar Javier Solares Orozco

MANUAL TECNICO

PROYECTO: ANALIZADOR LEXICO Y SINTACTICO

Auto: Erwin Alejandro Garcia Barrera
Carnet: 201700801
Versión: 1.00
Fecha: 27/10/18

Control de modificaciones

Control de versiones		
Versión	Descripción/Motivo	Fecha de presentación
1.00	Documento inicial	20/9/18

INDICE

Introducción

El programa consiste en realizar un laberinto con sus instrucciones indicadas mediante un texto con un lenguaje definido mediante el análisis léxico y sintáctico de este, generando la tabla de símbolos y de errores en caso de existir.

Objetivos

Se pretende mostrar de una manera clara y concisa el funcionamiento de la aplicación, obtener el laberinto y las tablas de símbolos y errores.

- Representar la funcionalidad técnica de la estructura, diseño y definición del aplicativo.
- Detallas la especificación de los requerimientos de hardware y software necesarios para la instalación de la aplicación.
- Describir las herramientas utilizadas para el diseño y desarrollo del programa.

Requerimientos para el uso de la aplicación

- Configuración de la pantalla con resolución de 1024x720 pixeles o superior.
- Mínimo un gigabyte de memoria RAM.
- 50 mb de espacio disponible.
- Sistema operativo Windows 7 o superior.

Herramientas utilizadas para el desarrollo

- Lenguaje C#
- Visual Studio Versión 2017
- Aplicación de Windows Forms en C#

Descripción de métodos

- Evento al presionar el botón analizar

Código	Descripción
String entrada;	Variable que recoge el texto del área de texto
AnalizadorLexico lex = new AnalizadorLexico();	Instancia de la clase analizador lexico
List<Token> lTokens = lex.escanear(entrada);	Crear la lista de tokens con el método escanear de la clase analizador lexico
lex.imprimirLista(lTokens)	Utilizar el método imprimir de la clase analizador lexico
Ciclo if else	Vericar si el texto contiene errores léxicos

- Método para pintar el texto

Código	Descripción
palabrasReservadas[…]	Arreglo que contiene las palabras aceptadas por el lenguaje
Int inicio	Variable que almacena el inicio del texto
Int final	Variable que almacena el final del texto
Ciclo for	Ciclo que recorre todo el texto
Ciclo if… else if… else	Ciclo que verifica si una palabra en el texto coincide con las palabras reservadas y aplica el color respectivo

Clase Token.cs: Clase que define los tokens aceptados por el lenguaje

Código	Descripción
Tipo tipoToken	Variable que define el tipo de token
String valor	Variable que contiene el valor del token
Int fila	Variable que contiene la fila en donde se encuentra el token
Int columna	Variable que contiene la columna donde se encuentra el token
Getvalor()	Método que retorna el valor del token

Gettipo()	Método que retorna el tipo
GetFila	Método que retorna la fila
GetColumna	Método que retorna la columna

Clase Error: clase que almacena los errores encontrados en el texto y no son reconocidos por el lenguaje

Código	Descripción
Char valor	Variable que define el valor del error
Int fila	Variable que contiene la fila en donde se encuentra el error
Int columna	Variable que contiene la columna donde se encuentra el error
Getvalor()	Método que retorna el valor del error
GetFila	Método que retorna la fila
GetColumna	Método que retorna la columna

Clase AnalizadorLexico.cs: clase que contiene la lógica detrás del analizador léxico utilizando el autómata finito determinista detallado más adelante.

Código	Descripción
List<Token>	Lista de tokens
List<Error>	Lista de errores
Int estado	Variable que contiene el estado del automata
String auxlex	Variable que contiene el Caracter leido
Int numerror	Variable que almacena el numero de errores
Int num	Variable que almacena el numero de tokens
Int fila	Variable que almacena la fila donde se encuentra el token
Int columna	Variable que alcana la columna donde se encuentra el token
Char c	Variable que almacena cada carácter del texto
Escanear	Método que recorre el texto de izquierda a derecha y de arriba hacia abajo
Ciclo for	Recorre cada carácter del texto
Ciclo switch	Contiene un caso para cada estado del automata

Ciclo if…. If else… else	Verifica que tipo de carácter se lee
addToken	Método que añade un token a la lista
Add Error	Método que añade un error a la lista
Imprimir lista	Metodo que crea una tabla que contendrá la información de los tokens
Ciclo for…each	Por cada token de la lista lo añade a una nueva fila en la tabla de tokens
Imprimir error	Método que crea una tabla de errores
Ciclo for…each	Por cada error lo añade a una nueva fila en la tabla de errores
Ciclo for	Por cada columna y fila añade los tokens a las tablas

Autómata utilizado

Expresión Regular

$$[S+|d+|o*|L+(-L+D)*]\#$$

Método del Árbol

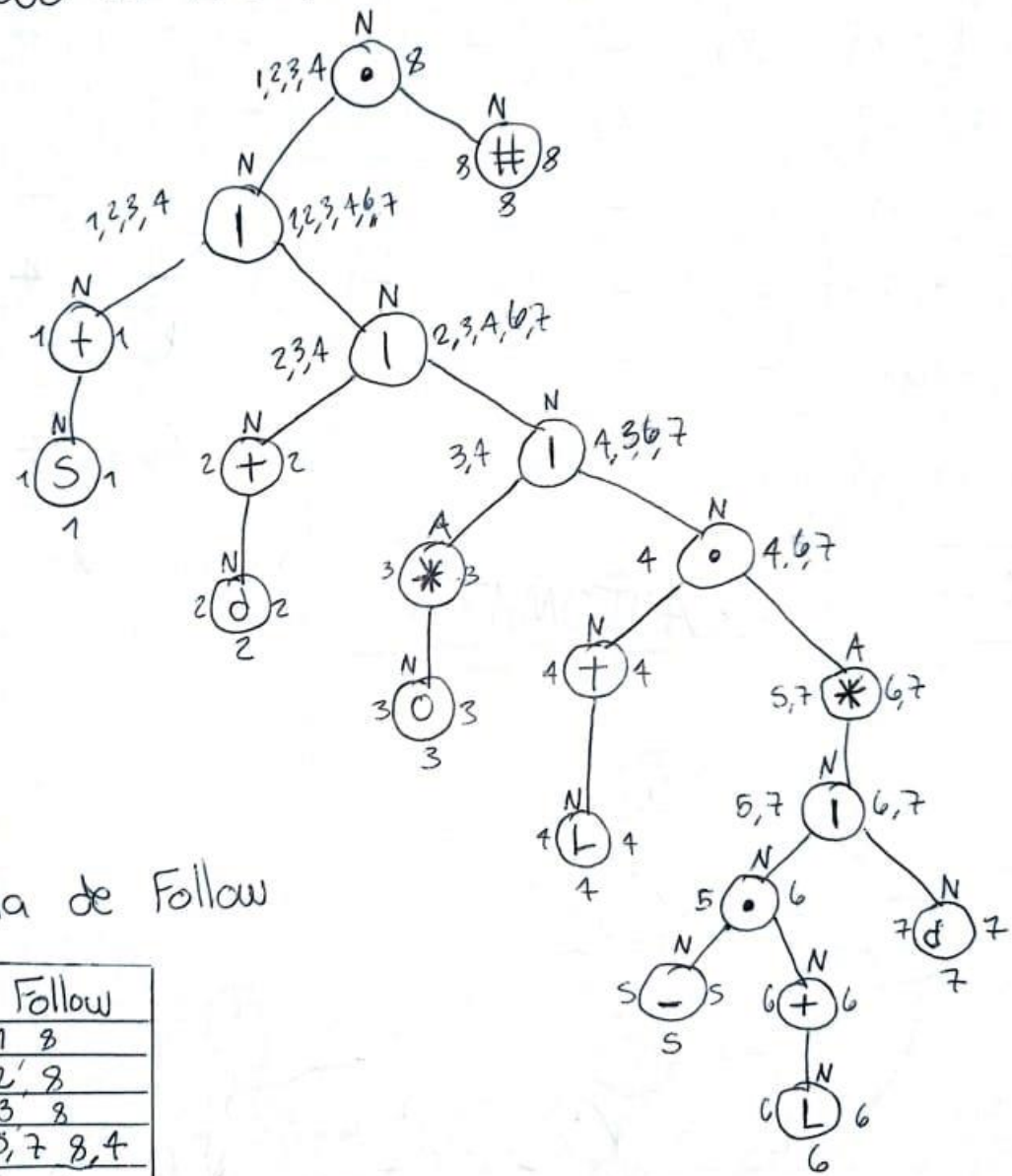
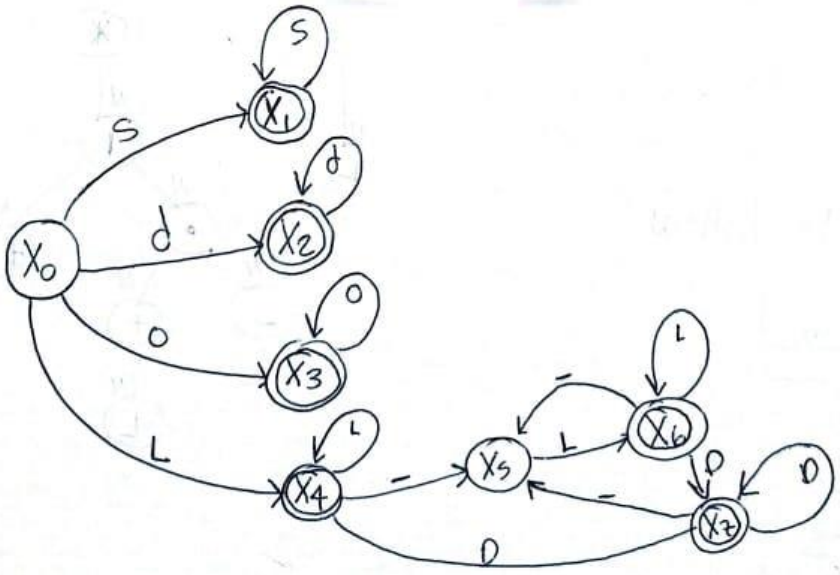


Tabla de Follow

No	Follow
1	1 8
2	2, 8
3	3 8
4	5, 7 8, 4
5	6,
6	6, 5, 7, 8
7	5, 7 8
8	—

• TABLA DE TRANSICIONES

	Estados \ Σ	1 S	2 d	3 O	4 L	5 _	6 L	7 D
	$X_0 = \{1,2,3,4\}$	X_1	X_2	X_3	X_4	–	–	–
*	$X_1 = \{1,8\}$	X_1	–	–	–	–	–	–
*	$X_2 = \{2,8\}$	–	X_2	–	–	–	–	–
*	$X_3 = \{3,8\}$	–	–	X_3	–	–	–	–
*	$X_4 = \{4,5,7,8\}$	–	–	–	X_4	X_5	–	X_7
	$X_5 = \{6\}$	–	–	–	–	–	X_6	–
*	$X_6 = \{5,6,7,8\}$	–	–	–	–	X_5	X_6	X_7
*	$X_7 = \{5,7,8\}$	–	–	–	–	X_5	–	X_7



Gramática tipo 2 utilizada

<A>------------------>[Principal]:{<B>}

<B>------------------>[Intervalo]:(num)<C>;

<C>------------------><BN>

|<BE>

|<BP>

|E

<BN>------------------>[Nivel]:{<BNC>}<C>

<BNC>----------------><D><IP><US><P>

<D>------------------>[Dimensiones]:(num,num);

<IP>----------------->[Inicio_personaje]:(num,num);

<US>----------------->[Ubicacion_Salida]:(num,num);

<P>------------------>[Pared]:{<E>}

<E>------------------><Casilla>

|<VariasCasillas>

|<Variable>

|<Asignacion>

|E

<Casilla>------------>[Casilla]:(num,num);<E>

<VariasCasillas>------>[Varias_Casillas]:(<Opciones>);<E>

<Variable>------------>[Variable]:<I>;<E>

<Asignacion>---------->identificador:=<J>;

<J>------------------>identificador

|num

|<Expresion>

<Expresion>---------><Opciones3>operador<Opciones3>

<Opciones3>--------->identificador

|num

<I>---------->identificador

|,<I>

<Opciones>---------->num..num,num

|num,num..num

|num..num,num..num

<BE>---------->[Enemigo]:{<F>}<C>

|E

<F>---------><Caminata>

|E

<Caminata>---------->[Caminata]:(<Opciones2>);<F>

<Opciones2>---------->num..num,num

|num,num..num

<BP>---------->[Personaje]:{<G>}<C>

<G>---------><Paso>

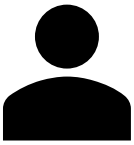
|<Caminata2>

|E

<Paso>---------->[Paso]:(num,num);

<Caminata2>---------->[Caminata]:(<Opciones2>);<G>

Contacto



Autor: Erwin Alejandro Garcia Barrera

Identificación: 201700801

e-mail: alejandro76.gb@gmail.com