

Desarrollo y Análisis de un Scanner en Lenguaje C--

Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Chihuahua

**Profesor**

Dr. Rodolfo Castelló Zetina

**Clase**

Diseño de Compiladores

**Alumno**

Sebastian Salazar Villanueva

A01568158

**Fecha**

2 de Abril del 2022

**Índice**

**1.** **Introducción**. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . .**3**

1.1. Resumen. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.2. Notación. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3

**2.** **Análisis**. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .**4**

2.1. Requerimientos . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .4

2.2. AFD . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .4

2.3. Tabla de Transiciones . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .5

2.4. Tabla de Símbolos. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .5

2.5. Mensajes de Error . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .5

**3.** **Diseño**. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .**6**

3.1. Pseudocódigo . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .7

3.2. Diagrama de Flujo . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .7

3.3. AFD . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .9

3.4. Lista de Tokens . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .9

3.5. Tabla de Símbolos . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

**4.** **Implementación** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **9**

# 

# 

# 

# 

# **1.** **Introducción**

## **1.1.** **Resumen**

En esta sección se proporcionará una introducción a todo el documento de Especificación de Requisitos Software(ERS). Consta de varias subsecciones: propósito, ámbito del sistema, definiciones, referencias y visión general del documento.

En este proyecto se dará a conocer el desarrollo y la entrega de un analizador léxico, también conocido como escáner, el cual se desarrollará a partir de los requerimientos, tablas de tokens, Automatas Finitos Deterministas y reglas del lenguaje C--. El lenguaje fue otorgado por el Dr. Rodolfo Castelló Zetina quien imparte la clase de Diseño de Compiladores. Usando el conocimiento y aprendizajes que se le otorgó a su servidor en todos los semestres que cursó la carrera de Tecnologías computacionales del Tecnológico de Monterrey, junto a herramientas que el Dr. Castello entregó para su desarrollo, el proyecto deberá ser completado con éxito, demostrando y aplicando así todos los conocimientos que se obtuvieron en la realización del escáner.

## **1.2.** **Notación**

Máquinas de Estados Finitos: es un modelo abstracto de la computación que es usado para modelos lógicos. La única forma de que un lenguaje sea considerado como regular es que este pueda ser reconocido por una máquina de estados finitos.

Expresiones Regulares: Cadena de caracteres que se utiliza para describir patrones dentro de otros strings, usando tanto delimitadores como reglas de sintaxis.

Tabla de transiciones: Tabla que realiza la representación de un AFD(Autómata finito determinista), la cual está conformada por:

* Filas: Representan los distintos estados.
* Columnas: Representan los símbolos de entrada.
* Entradas: Representan el siguiente estado a dirigir.

El modelo que se utilizó para la realización del escáner fue el de “Table-Driven”(Basado en Tablas). El modelo Table-Driven se usa normalmente para un acercamiento más preciso a una ingeniería de Desarrollo de Software que apunta a simplificar y generalizar aplicaciones separándolas del control de variables y de las reglas de parámetros del código y colocando estos en una parte externa separados de la lógica del código.

Para la fase de análisis se concluyó que este modelo era mucho mejor de implementar debido a que se tenía una estructura mucho más viable, separada y sencilla para adaptarse a cambios en caso de que estos fueran necesarios.

Para la fase de diseño fue mucho más sencillo, pues si bien es importante crear el diseño del código, es aún más importante saber lo que se va a hacer antes de este. La fase de diseño fue una de las más importantes y que se le dio más prioridad, al realizar el AFD(Autómata finito determinista) se logró ver un panorama más claro de lo que el escáner debía de manejar y cuando iba a hacerlo, ya que al crear este autómata se pudieron ver errores comunes y soluciones sencillas para su implementación.

Para la fase de Desarrollo, de la misma manera, fue más sencillo de implementar gracias a que se obtuvieron las bases con el análisis y el diseño, después de tener estas dos fases completadas era más que implícito como el código debería de ser escrito y la forma en la que debía adaptarse a el lenguaje usado, siendo en este caso el lenguaje C++.

Después de revisar las obvias ventajas que tenía el modelo “Table-Driven”, este fue elegido por todas estas y por llevar un seguimiento más sólido por si el escáner debía adaptarse a cambios. De no ser un modelo “Table-Driven”, sería muy difícil adaptarse a cambios y sobre todo darle un mejor seguimiento en caso de ser necesario.

El programa que se eligió para el desarrollo fue C++ debido a las razones siguientes:

* La facilidad del manejo de datos, tanto crearlos como usarlos es de mucha utilidad para la implementación.
* Es más rápido a comparación de otros lenguajes.
* Es orientado a objetos, lo cual da una facilidad al trabajar con estados en la tabla de transiciones de su misma implementación.
* Durante el periodo de su servidor(Sebastian Salazar) en la carrera de Tecnologías Computacionales en el Tecnológico de Monterrey, se vieron muchos lenguajes de programación. Sin embargo, este es en el que el alumno se siente mucho más cómodo trabajando y, el cual, le tiene más cariño por ser el primer lenguaje de programación aprendido.
* El manejo de memoria es más sencillo.
* La manera de manejar los argumentos en las funciones.

# **2.** **Análisis**

## **2.1.** **Requerimientos**

* El sistema deberá de realizarse en el lenguaje de programación C++.
* El sistema deberá reconocer e identificar los tokens o líneas de caracteres en el orden en el que aparecen en el archivo fuente y que sean posibles de identificar en el lenguaje C--.
* El sistema deberá construir la versión preliminar de la tabla de símbolos para todo tipo de tokens existentes
* El sistema contendrá las siguientes “Keywords”, las cuales se consideran como palabras clave en español, son palabras reservadas y no hacen distinción entre mayúsculas y/o mayúsculas:

**-else**

**-if**

**-int**

**-return**

**-void**

**-while**

**-input**

**-output**

* El sistema contendrá los siguientes símbolos especiales:
* **Operación de Suma Aritmética**

**- Operación de Resta Aritmética**

**\* Operación de Multiplicación Aritmética**

**/ Operación de División Aritmética**

**< Operador Lógico Menor Que**

**<= Operador Lógico Menor o Igual Que**

**> Operador Lógico Mayor Que**

**>= Operador Lógico Mayor o Igual Que**

**== Operador Lógico igual**

**!= Operador Lógico Diferente**

**= Asignación**

**; Punto y Coma**

**, Coma**

**( Paréntesis Abierto**

**) Cerrar Paréntesis**

**[ Corchetes Abiertos**

**] Cerrar Corchetes**

**{ Abrir Corchetes**

**} Cerrar Corchetes**

**/\* Abrir Comentario**

**\*/ Cerrar Comentario**

* Otros tokens son “**ID**” y “**NUM**”, las Expresiones Regulares correspondientes son las siguientes:

**ID = letra+**

**NUM = dígito+**

**letra = [a-z A-Z]**

**dígit0 = [0-9]**

* Los Identificadores son inestables ante las letras, las letras mayúsculas y minúsculas son distintas.
* El espacio blanco consiste en “**blanks**” (espacios en blanco),”**newlines**”(indican final de línea y salto a la siguiente) y **tabs.** El espacio en blanco es ignorado, pero deberá ser reconocido. El espacio en blanco junto con “**ID’s**”, “**NUM’s**” y “**keywords**”, son considerados como **delimitadores**.
* Los comentarios serán estilo a lenguaje C, son puestos por /\* … \*/. Los comentarios pueden ser colocados en donde sea donde pueda aparecer un espacio en blanco. Los comentarios no pueden ser colocados entre o dentro de tokens. Los comentarios pueden incluir más de una línea.
* En caso de errores, estos deberán especificarse para así tener una mejor comprensión para corregir el error que despliega el sistema.
* Lista de Tokens Validos:

1. **RETURN**
2. **WHILE**
3. **VOID**
4. **ELSE**
5. **IF**
6. **INT**
7. **INPUT**
8. **OUTPUT**
9. **>**
10. **<**
11. **=**
12. **<=**
13. **==**
14. **>=**
15. **!=**
16. **(**
17. **)**
18. **[**
19. **]**
20. **{**
21. **}**
22. **/**
23. **\***
24. **-**
25. **+**
26. **,**
27. **;**
28. **Identificador**
29. **Constante Entera**

* Conjunto de delimitadores

Los delimitadores deberán depender del estado en el que se encuentre actualmente el autómata, pues si bien se podría definir una lista fija de delimitadores, esta sería incorrecta debido a que los delimitadores cambian conforme se avanza en el AFD.

* Se deberá realizar la implementación de un AFD dentro del sistema con el modelo de “Table-Driven” implementado para un mejor uso de datos.
* Se deberá realizar la implementación de una Tabla de Transiciones la cual deberá depender de la estructura del AFD para dar a conocer una estructura completa respecto a cada estado y a donde va a avanzar.
* Se deberá especificar dentro del sistema un mensaje con los tipos de error y porque estos están siendo llevados a cabo, es decir, la razón por la cual ocurrió ese error.
* Se deberán guardar las tablas de símbolos en dos listas diferentes siendo estas una para números y otra para identificadores.
* Se deberá guardar el resultado de la salida en una lista para mantener la información archivada en esta misma.
* Se deberá de tomar valor de la entrada con la lectura de un archivo.

## **2.2.** **AFD**

## 

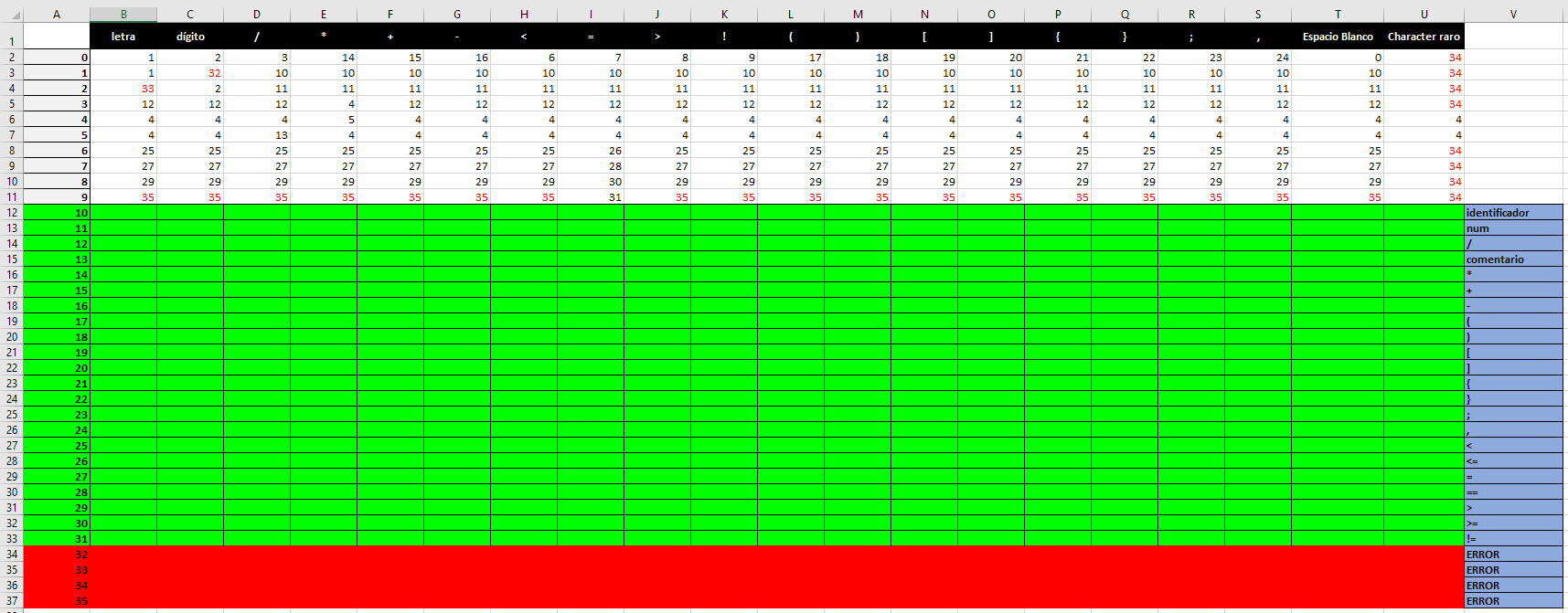
**Figura 1.** AFD Lenguaje C--

En la Figura 1 se puede observar el AFD, siendo este el Autómata Finito Determinista del lenguaje C--.Un AFD consiste en un Autómata que solo tiene un solo camino a cada estado finito, es decir, solo existe una forma de llegar desde un estado a otro. Este está conformado por líneas que representan la transición de un estado a otro, círculos de una sola línea representando estados no aceptores y círculos de doble línea representando estados aceptores. Las especificaciones de las transiciones se representan a un lado de la línea ya mencionada y el número de estados se representa dentro de los círculos de cada estado.

Antes de enumerar los estados, se creó una estructura en el diagrama en donde se obtienen los caminos de transición del AFD dependiendo de los símbolos especiales, letras o números y finalmente después de obtener las transiciones se determinan los estados aceptores y de error.

Se comenzó a numerar por el estado 0 como estado de partida a cualquier camino que el lenguaje pudiera tomar, continuando con los demás estados no aceptores. Siguiendo después de los no aceptores, se enumeraron los aceptores y por último los errores, tomándose en cuenta sólo 4 de estos.

## **2.3.** **Tabla de Transiciones**



La Tabla de Transiciones, como explicado anteriormente, se retomó con columnas representando estas mismas los caracteres de entrada y las filas representando los distintos estados siendo un total de 35. Las filas de color blanco representan los estados donde aun es posible avanzar en el AFD, en cuanto al contenido dentro de las celdas representan el estado al que se avanzará. Si el color del estado dentro de cada celda es rojo, significa que este estado próximo es un estado de error.

En cuanto a las filas en verde representan los estados aceptores y las rojas los estados de error. Por último la columna final fuera de las transiciones de estados representa el tipo de estado en el que se encuentran las filas de estos.

## **2.4.** **Tablas de Símbolos**

Las Tablas de Símbolos son estructuras de datos que se usan por el compilador para mantener información, se representaron en 2 tablas distintas ya anteriormente mencionadas, siendo una de números y otra de identificadores. Los números representan, como su nombre lo indica, números que se usaron de entrada y los identificadores como conjunto de caracter o caracteres juntos.

## **2.5.** **Mensajes de error**

Los mensajes de error fueron un total de 4, se especifican en los últimos 4 estados del AFD como:

Error 1: “------ERROR, dígito después de letra-----”

Error 2: “------ERROR, letra después de dígito------”

Error 3: “ ----ERROR, caracter no valido encontrado----”

Error 4: “-----ERROR, no se encontró '=' después de '!'-----”

# **3.** **Diseño**

## **3.1.** **Pseudocódigo**

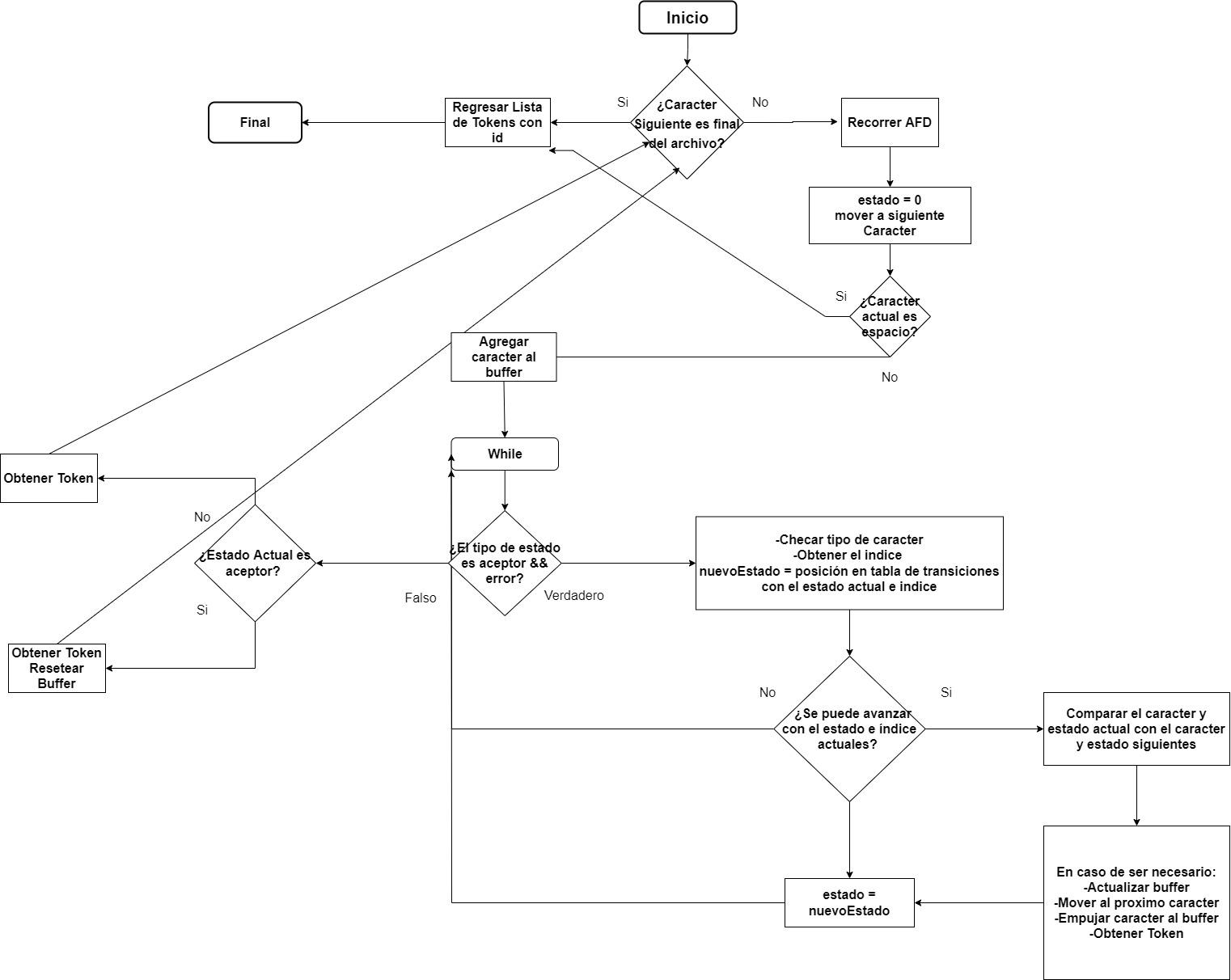
| // Primer método  checarCaracter(){  if(aceptaCaracter == false)  checaLetra(caracterActual)  if(aceptaCaracter == false)  checaNumero(caracterActual)  if(aceptaCaracter == false)  checaCaracterEspecial(caracterActual)  if(aceptaCaracter == false)  checaEspacios(caracterActual)  // Dentro de cada chequeo individual se transforma la llave de  // aceptarCaracter a verdadero en caso de que se encuentre y  // de esta forma no tener que recorrer los faltantes  }  // Segundo método  guardarymostrarToken(){  if(tokenNum == 0)  // Ignora el comentario  else if(0<tokenNum < 28)  insertarenLista();  else if(tokenNum == 28)  insertarenlistaLetra(); // Uso de Tablas de Simbolos Letra  insertarenLista(); // Uso de lista para la salida  else if(tokenNum == 29)  insertarenlistaNumero(); // Uso de Tablas de Simbolos Numero  insertarenLista(); // Uso de lista para la salida  }  // Tercer método  estadoRecorridoAFD(){  estado = 0;  siguienteCaracter;  // Avanza al siguiente caracter  // Se obtiene el índice del caracter  while (!Aceptado[estado] && !Error[estado] ) {  nuevoEstado = tablaTransiciones[estado, índiceCaracter];  if (Avanza[estado, índiceCaracter])  siguienteCaracter;  estado = nuevoEstado;  }  if (Aceptado[estado] )  guardarymostrarToken;  else  }  main(){  while(siguienteCaracter != EOF) // EOF—> fin del archivo  estadoRecorridoAFD();    imprimirListaSalida(); // Regresar la lista con su salida esperada    } |
| --- |

En esta sección se muestran 3 métodos distintos y el método main en pseudocódigo siendo estas las más importantes y las que más destacan en cuanto a funcionalidad. El primer método mostrado es para revisar el tipo de caracter que es, teniendo la posibilidad de ser letra, numero, espacio o caracter especial. El segundo método hace referencia a cómo se van a guardar los tokens en sus respectivas listas para mostrarlo en la salida.

El tercer método está basado en la implementación del Dr. Castelló de la presentación del Análisis Léxico analizada en la clase impartida de Diseño de Compiladores. Este método es específicamente desarrollado para que el AFD pueda proyectarse de una manera más estructurada y convencional, ya que, al momento de querer realizar uno o más cambios este sería mucho más sencillo de editar.

Por último se encuentra el main, el cual se encarga de que se realice todo el ciclo del AFD para finalmente obtener la salida deseada como resultado.

## **3.2.** **Diagramas de Flujo**



**Figura 2.** Diagrama de Flujo

Como se puede observar en la Figura 2 se encuentra el diagrama de flujo del AFD para su implementación en código. Cabe destacar que se eligió este diagrama debido a que desde el inicio hasta el final del sistema el algoritmo utilizado corresponde al AFD, el cual contiene todo lo que se necesita para obtener la entrada y darle salida a un resultado exitoso y esperado.

## **3.3.** **AFD**

El autómata finito determinista se representó de la siguiente forma:

**Arreglos de caracteres**

caracterActual = ' ';

letraArr[52] = { 'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z','A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z' };

numArr[10] = { '0','1','2','3','4','5','6','7','8','9' };

caracEspecialesArr[16] = { '/',';',',','=','\*','+','-','(',')','<','>','[',']','{','}', '!' };

espaciosArr[4] = { '\n','\t', ' ', '\0' };

**Manejo de estados, estados aceptores, de error y los que pueden avanzar**

bool aceptarEstados[36] = { false, false, false, false, false, false, false, false, false, false, true, true, true, true, true, true, true, true, true, true,true, true, true, true, true, true, true, true, true, true, true, true, false, false, false ,false };

bool errorEstados[36] = { false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false,true, true,true, true };

avanEstados[10][20] = {

{ true, true, true, false, false, false, true, true, true, true, false, false, false, false, false, false, false, false, true, false},

{ true, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false,false},

{ false, true,false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false,false},

{ false, false,false, true, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false,false},

{ true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true},

{ true, true, false, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true},

{ false, false,false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false,false},

{ false, false,false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false,false},

{ false, false,false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false,false},

{ false, false,false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false,false},

};

**Tabla de Transiciones**

tablaTrans[10][20] = {

{1, 2 , 3, 14, 15, 16, 6, 7, 8, 9, 17, 18, 19, 20, 21, 22, 23, 24, 0, 34},

{1, 32, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 34},

{33, 2, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 34},

{12, 12, 12, 4, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 34},

{4, 4, 4, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4},

{4, 4, 13, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4},

{25, 25, 25, 25, 25, 25, 25, 26, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 34},

{27, 27, 27, 27, 27, 27, 27, 28, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 34},

{29, 29, 29, 29, 29, 29, 29, 30, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 34},

{35, 35, 35, 35, 35, 35, 35, 31, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 34}

};

El manejo de caracteres, tipos de estado y tabla de transiciones se desarrollaron en un arreglo dependiendo del tipo de dato que requerían. Pues si bien tanto la tabla de transiciones como el arreglo de números se manejaron con el tipo de dato entero, los demás arreglos se manejaron con tipo de dato caracter mientras que los tipo de estado se observó más conveniente desarrollar con tipo de dato booleano.

## **3.4.** **Lista de Tokens**

lista\_token[27] = { "RETURN", "WHILE", "VOID", "ELSE", "IF", "INT", "INPUT" , "OUTPUT" , ">" , "<" , "=", "<=" , "==", ">=" , "!=", "(", ")" , "[" , "]" , "{" , "}" , "/" , "\*", "-" , "+", "," , ";" };

La lista de tokens, como la mayoría de estructuras en el sistema, se implementó como un arreglo, comparando el elemento que se encuentra dentro de cada posición con su posición misma.

## **3.5.** **Tabla de Símbolos**

vector<int> vecNum;

vector<string> vecLetra;

# **4.** **Implementación**

# **5.** **Verificación y Validación**

REFERENCIAS:

[What is Table-Driven Design? - Definition from Techopedia](https://www.techopedia.com/definition/30408/table-driven-design#:~:text=Table%2DDriven%20Design-,What%20Does%20Table%2DDriven%20Design%20Mean%3F,those%20in%20separate%20external%20tables.)

[Finite state machines — Isaac Computer Science](https://isaaccomputerscience.org/concepts/dsa_toc_fsm?examBoard=all&stage=all)

[Tabla de transición en autómatas – Acervo Lima](https://es.acervolima.com/tabla-de-transicion-en-automatas/)

TC3048\_Chapter\_2\_Lexical\_Analysis.pdf