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

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 Válidos:

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(vectores) 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

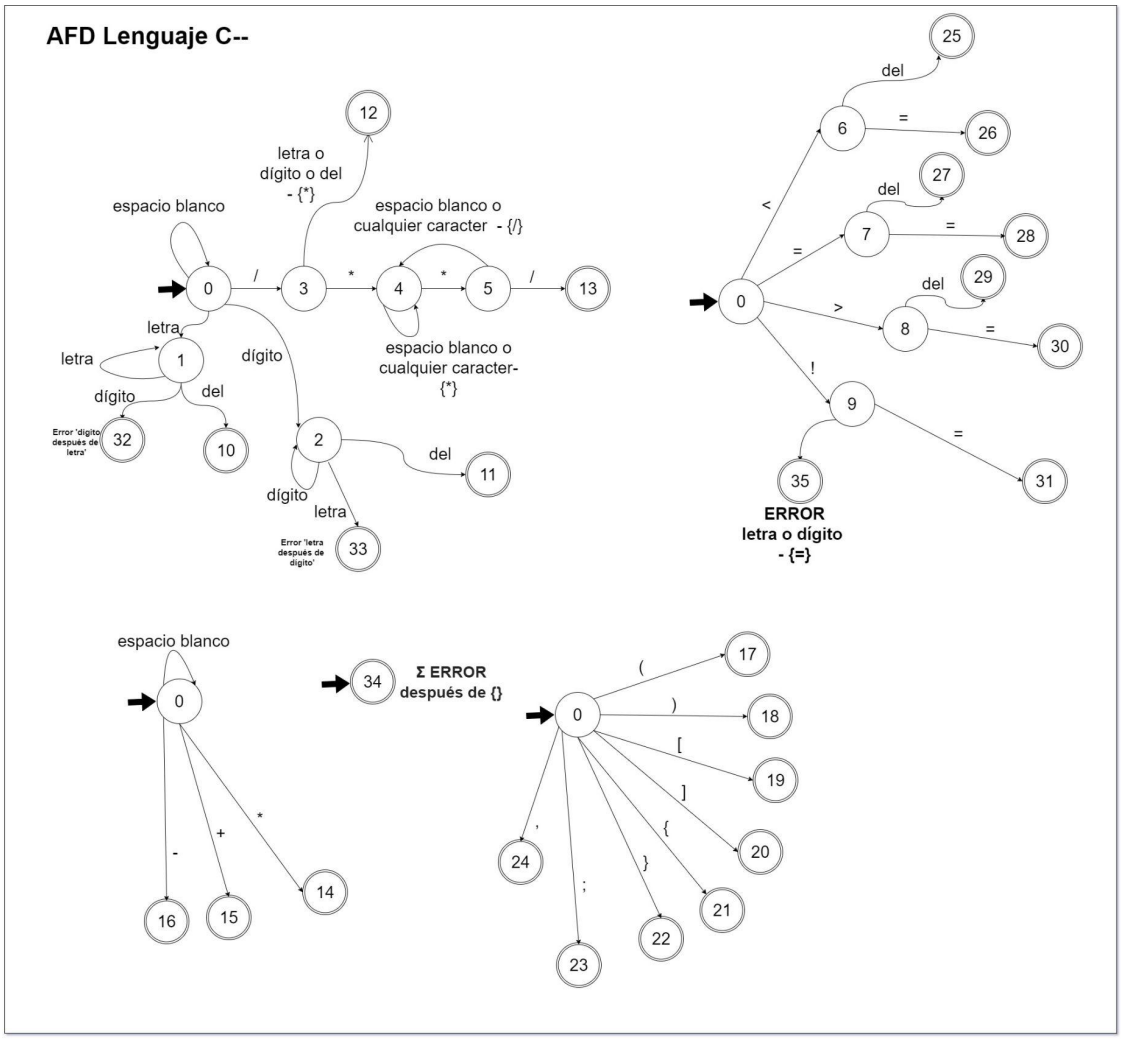


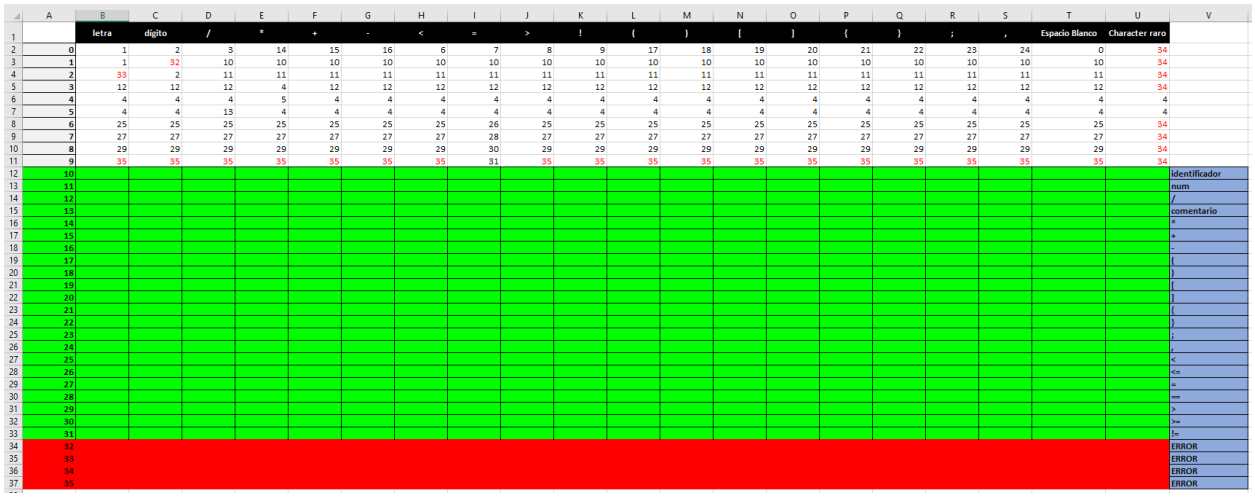
Figura 1. AFD Lenguaje C--

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



	letra	dígito	/	*	+	-	<	=	>	!	(	)	[	]	{	}	;	,	Espacio Blanco	Character raro	
0	1	2	3	14	15	16	6	7	8	9	17	18	19	20	21	22	23	24	0	34	
1	1	32	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	34	
2	33	2	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	34	
3	12	12	12	4	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	34	
4	4	4	4	5	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	
5	4	4	13	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	
6	25	25	25	25	25	25	25	26	25	25	25	25	25	25	25	25	25	25	25	34	
7	27	27	27	27	27	27	27	28	27	27	27	27	27	27	27	27	27	27	27	34	
8	29	29	29	29	29	29	29	30	29	29	29	29	29	29	29	29	29	29	29	34	
9	35	35	35	35	35	35	35	31	35	35	35	35	35	35	35	35	35	35	35	34	
10																					identificador
11																					num
12																					/
13																					comentario
14																					*
15																					+
16																					-
17																					(
18																					)
19																					[
20																					]
21																					{
22																					}
23																					;
24																					,
25																					<
26																					<=
27																					=
28																					==
29																					>
30																					>=
31																					!=
32																					ERROR
33																					ERROR
34																					ERROR
35																					ERROR

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

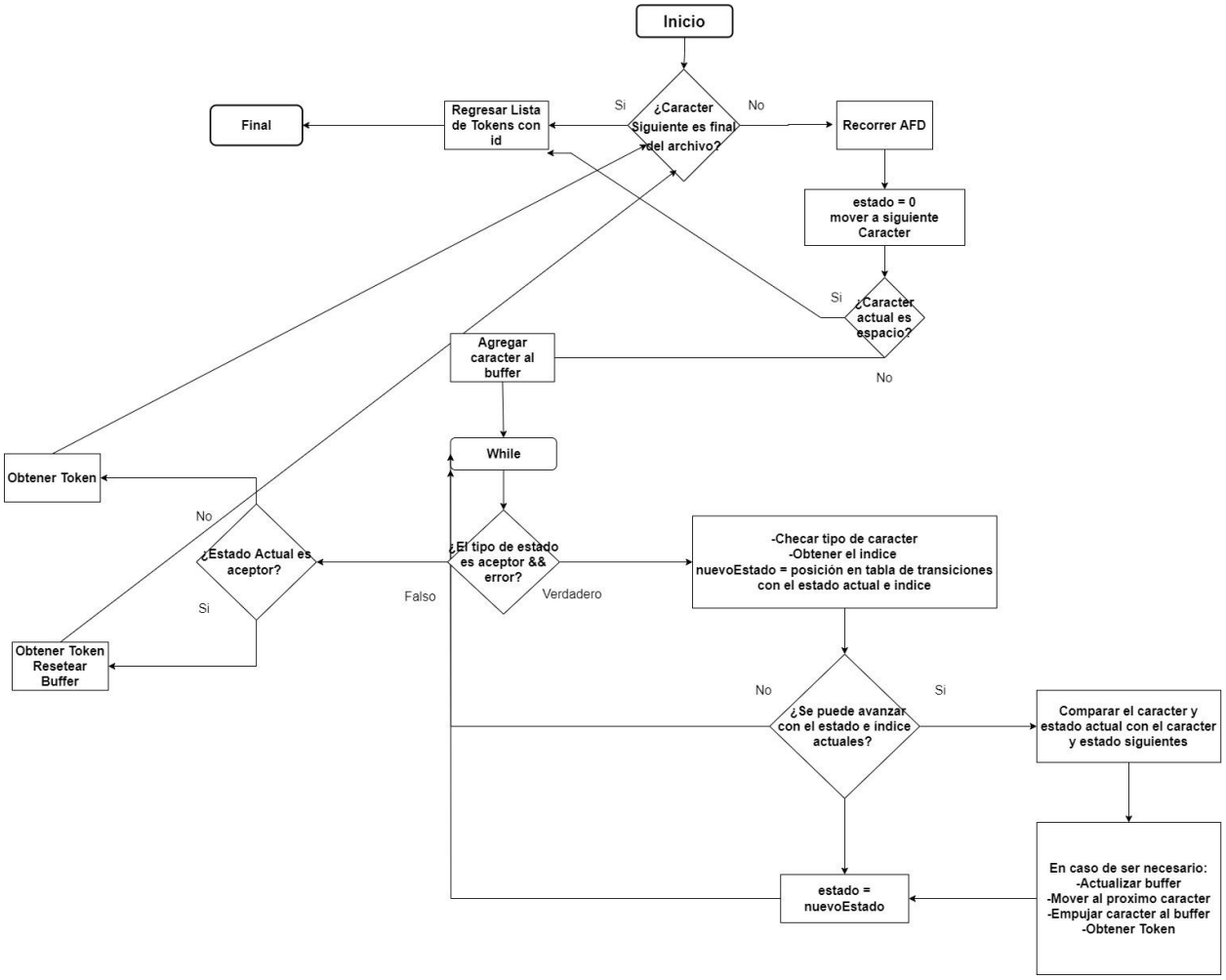


Figura 2. Diagrama de Flujo

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;
```

Las tablas de símbolos también fueron implementadas con arreglos, pero esta vez fueron arreglos que podrían cambiar su tamaño llamados vectores. La razón por la que se utilizaron vectores fue por la manera en la que se pueden insertar, expandir y seleccionar objetos dentro de los arreglos, lo cual ayudaba mucho a el manejo de datos al momento de insertarlos en un vector único para la salida esperada.

4. Implementación

4.1. Código Fuente

```cpp
// Scaner_A01568158.cpp
// Table-Driven
// Hecho por Sebastian Salazar Villanueva
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

// Variables Globales
fstream fin("Sample.txt", fstream::in); // Modo lectura File in
bool aceptchar; // Llave para saber si un caracter es letra, numero o caracter especial, de esta forma no pasar por los demás
string identifier; // Buffer donde se guarda cada uno de los caracteres ya sean
int characterType; // Indica si es letra, numero o caracter especial

// El caracter actual que se lee y los Arreglos de Caracteres
char caracterActual = ' ';
char 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',
int numArr[10] = { '0','1','2','3','4','5','6','7','8','9' };
char caracEspecialesArr[16] = { '/',',','.','=','*','+','-','(',')','<','>','[',']','{','}', '!' };
char espaciosArr[4] = { '\n','\t', ' ', '\0' };

// Lista de Tokens
string lista_token[27] = { "RETURN", "WHILE", "VOID", "ELSE", "IF", "INT", "INPUT" , "OUTPUT" , ">" , "<" , "=", "<=" , "==", ">=" , "!=", "(", ")" , "[" , "]" , "{" , "}" , "/" , "*", "-" , "+", "," , ";" };

// Llave para saber que tipo de caracter es el caracter actual
bool esLetra = false;
bool esNum = false;
bool esEspecial = false;
bool esEspacio = false;

// Estados, se especifica cuales estados son aceptores, de error, un arreglo bidimensional para revisar si puede avanzar
bool aceptarEstados[36] = { 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, tru
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,
bool 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},
};
```

```cpp

//Tabla de transiciones
int 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}
};

namespace Vec{ ... }
// *Vectores con sus tipos de datos*
Vec::vector<string> vecOutput; // Vector para la salida
Vec::vector<int> vecNum; // Vector para los números
Vec::vector<string> vecLetra; // Vector para los identificadores

// Transforma todas las letras de un identificador para revisar si es una Keyword
string upperCase(string ident) {
    string upper = "";
    for (int i = 0; i < ident.length(); i++) {
        upper += toupper(ident[i]);
    }

    return upper;
}

// Compara en que estado se encuentra y regresa Token
int compararToken(int estado) {

    string upper = upperCase(identifier);
    switch (estado) {
    case 10:
        for (int d = 0; d <= 7;) {
            if (upper == lista_token[d]) {
                return d + 1;
            }
            d++;
        }
        return 28;
        break;
```

```
namespace Vec {
    template<typename Id>
    class vector {
    private:
        // La capacidad inicial del arreglo con los 29 tokens
        //size_t tipo de entero sin ser asignado
        static const size_t capacidad_inicio = 1;
        // La capacidad actual del arreglo
        size_t capacidad_actual;
        // El num_obj actual del arreglo
        size_t num_obj;
        //El array que contiene la data
        Id* arr_datos;
    public:
        //Constructor
        vector<Id>() : capacidad_actual(capacidad_inicio), arr_datos(new Id[capacidad_inicio]), num_obj(0) { }
        // Regresa el numero de objetos que estan insertados, los cuales sirven para recorrer todo el vector de output
        size_t getActualPos() { ... }
        // Obtiene lo que se encuentra en la posición especificada en el parámetro
        Id& getIdent(int x) {
            return  arr_datos[x];
        }
        // Expande el vector para que sea posible insertar
        void reservar(size_t capacidad_nueva) { ... }
        // Insertar objeto en el vector especificado dependiendo de su tipo de dato
        void insertar(size_t index, const Id& valor) { ... }
    };
}
// *Vectores con sus tipos de datos*
Vec::vector<string> vecOutput; // Vector para la salida
Vec::vector<int> vecNum; // Vector para los números
Vec::vector<string> vecLetra; // Vector para los identificadores
```

La implementación para la estructura de datos de vectores fue basada en el libro “OBJECTS, ABSTRACTION, DATA STRUCTURES AND DESIGN USING C++”, utilizado previamente en la clase de estructura de datos.

```
    case 11:
        return 29;
        break;
    case 12:
        return 22;
        break;
    case 13:
        return 0;
        break;
    case 14:
        return 23;
        break;
    case 15:
        return 25;
        break;
    case 16:
        return 24;
        break;
    case 17:
        return 16;
        break;
    case 18:
        return 17;
        break;
    case 19:
        return 18;
        break;
    case 20:
        return 19;
        break;
    case 21:
        return 20;
        break;
    case 22:
        return 21;
        break;
    case 23:
        return 27;
        break;
    case 24:
        return 26;
```

```
            break;
        case 25:
            return 10;
            break;
        case 26:
            return 12;
            break;
        case 27:
            return 11;
            break;
        case 28:
            return 13;
            break;
        case 29:
            return 9;
            break;
        case 30:
            return 14;
            break;
        case 31:
            return 15;
            break;
        case 32:
            cout << " ------ERROR, dígito después de letra----- ";
            abort();
            break;
        case 33:
            cout << " ------ERROR, letra después de dígito------ ";
            abort();
            break;
        case 34:
            cout << " ----ERROR, caracter no valido encontrado---- ";
            abort();
            break;
        case 35:
            cout << " -----ERROR, no se encontró '=' después de '!'-----";
            abort();
            break;


        }
    }
```

```
// Avanza a el proximo caracter a leer
void siguienteChar() {
    fin >> noskipws >> caracterActual;
}

// Regresa el indice en la tabla de transiciones dependiendo de si es numero, letra, caracter especial o espacio
void indicarChar() {

    if (esLetra == true) {
        characterType = 0;
    }
    else if (esNum == true) {
        characterType = 1;
    }
    else if (esEspecial == true) {

        if (caracterActual == '/') {
            characterType = 2;
        }

        else if (caracterActual == '*') {
            characterType = 3;
        }

        else if (caracterActual == '+') {
            characterType = 4;
        }

        else if (caracterActual == '-') {
            characterType = 5;
        }

        else if (caracterActual == '<') {
            characterType = 6;
        }

        else if (caracterActual == '=') {
            characterType = 7;
        }

        else if (caracterActual == '>') {
```

```
            else if (caracterActual == '>') {
                characterType = 8;
            }

            else if (caracterActual == '!') {
                characterType = 9;
            }

            else if (caracterActual == '(') {
                characterType = 10;
            }

            else if (caracterActual == ')') {
                characterType = 11;
            }

            else if (caracterActual == '[') {
                characterType = 12;
            }

            else if (caracterActual == ']') {
                characterType = 13;
            }

            else if (caracterActual == '{') {
                characterType = 14;
            }

            else if (caracterActual == '}') {
                characterType = 15;
            }

            else if (caracterActual == ';') {
                characterType = 16;
            }

            else if (caracterActual == ',') {
                characterType = 17;
            }
        }
        else if (esEspacio == true) {
```

```
            characterType = 18;
        }
        else {
            characterType = 19;
        }

    }

    // Checa si es una letra
    void checkLetra(char characters) {
        for (int i = 0; i <= 51;) {
            if (characters == letraArr[i]) {
                esLetra = true;
                aceptchar = true;
                i = 51;
            }
            i++;
        }
    }

    // Checa si es un Numero
    void checkNum(char characters) {
        if (aceptchar == false) {
            for (int x = 0; x <= 9;) {
                if (characters == numArr[x]) {
                    aceptchar = true;
                    esNum = true;
                    x = 9;
                }
                x++;
            }
        }
    }

    // Checa si es un caracter especial
    void checkSpecial(char characters) {
        if (aceptchar == false) {
            for (int y = 0; y <= 15;) {
                if (characters == caracEspecialesArr[y]) {
                    esEspecial = true;
                    aceptchar = true;
```

```
                y = 17;
            }
            y++;
        }
    }
}

// Checa si es un espacio
void checkEspacios(char characters) {
    if (aceptchar == false) {
        for (int y = 0; y <= 4;) {
            if (characters == espaciosArr[y]) {
                esEspacio = true;
                aceptchar = true;
                y = 3;
            }
            y++;
        }
    }
}

// Se encarga de checar que tipo de caracter es
void checkCharacter(char characters) {
    if (aceptchar == false) {
        checkLetra(characters);
    }
    if (aceptchar == false) {
        checkNum(characters);
    }

    if (aceptchar == false) {
        checkSpecial(characters);
    }

    if (aceptchar == false) {
        checkEspacios(characters);
    }
}

// Regresa que tipo de caracter es para luego ser usado como index en peekCharType
int getCharType(char characters) {
```

```

// Regresa que tipo de caracter es para luego ser usado como index en peekCharType
int getCharType(char characters) {

    if (characters == EOF) {
        return 3;
    }
    else {
        for (int i = 0; i <= 51;) {
            if (characters == letraArr[i]) {
                return 0;
            }
            i++;
        }
        for (int x = 0; x <= 9;) {
            if (characters == numArr[x]) {
                return 1;
            }
            x++;
        }

        for (int y = 0; y <= 15;) {
            if (characters == caracEspecialesArr[y]) {
                return 2;
            }
            y++;
        }

        for (int y = 0; y <= 5;) {
            if (characters == espaciosArr[y]) {
                return 3;
            }
            y++;
        }
    }
}

```

```
436     // Checa el tipo con argumento del siguiente caracter
437   int peekCharType(char c) {
438       if (getCharType(c) == 0) {
439           return 0;
440       }
441       else if (getCharType(c) == 1) {
442           return 1;
443       }
444
445       else if (getCharType(c) == 2) {
446
447           if (c == '/') {
448               return 2;
449           }
450
451           else if (c == '*') {
452               return 3;
453           }
454
455           else if (c == '+') {
456               return 4;
457           }
458
459           else if (c == '-') {
460               return 5;
461           }
462
463           else if (c == '<') {
464               return 6;
465           }
466
467           else if (c == '=') {
468               return 7;
469           }
470
471           else if (c == '>') {
472               return 8;
473           }
474
475           else if (c == '!') {
476               return 9;
```

```
            else if (c == '(') {
                return 10;
            }

            else if (c == ')') {
                return 11;
            }

            else if (c == '[') {
                return 12;
            }

            else if (c == ']') {
                return 13;
            }

            else if (c == '{') {
                return 14;
            }

            else if (c == '}') {
                return 15;
            }

            else if (c == ';') {
                return 16;
            }

            else if (c == ',') {
                return 17;
            }
        }
        else if (getCharType(c) == 3) {
            return 18;
        }
        else {
            return 19;
        }

    }

```

```

// Resetea las llaves de el caracter
void restartChar() {
    esLetra = false;
    esNum = false;
    esEspecial = false;
    esEspacio = false;
    aceptchar = false;
}

// Inserta el token, con su id(De ser necesario) en la el vector de output
void mostrarToken(int estado) {
    if (compararToken(estado) == 0) {
        //Ignorar comentario
    }
    else if(compararToken(estado) > 0 && compararToken(estado) < 28) {

        vecOutput.insertar(vecOutput.getActualPos(), to_string(compararToken(estado)));
    }
    else if (compararToken(estado) == 28) {
        bool existe = false;
        int x;
        for (x = 0; x < vecLetra.getActualPos();) {
            if (identifier == vecLetra.getIdent(x)) {
                existe = true;
                break;
            }
            x++;
        }
        if (existe) {
            vecOutput.insertar(vecOutput.getActualPos(), to_string(compararToken(estado)) + ',' + to_string(x));
        }
        else {
            int index = vecLetra.getActualPos();
            vecLetra.insertar(vecNum.getActualPos(), identifier);
            vecOutput.insertar(vecOutput.getActualPos(), to_string(compararToken(estado)) + ',' + to_string(index));
        }
    }
    else if (compararToken(estado) == 29) {
        bool existe = false;
        int x;
```

```
            bool existe = false;
            int x;
            for (x = 0; x < vecNum.getActualPos();) {
                if (identifier == to_string(vecNum.getIdent(x))) {
                    existe = true;
                    break;
                }
                x++;
            }
            if (existe) {
                vecOutput.insertar(vecOutput.getActualPos(), to_string(compararToken(estado)) +  ',' + to_string(x));
            }
            else {
                int index = vecNum.getActualPos();
                vecNum.insertar(vecNum.getActualPos(), stoi(identifier));
                vecOutput.insertar(vecOutput.getActualPos(), to_string(compararToken(estado)) + ',' + to_string(index));
            }
        }
    }

    // Algoritmo del DFA, el cual facilita el manejo del caracter actual
    void estadoRecorrido() {
        int estado = 0;
        siguienteChar();

        // Si el valor es un espacio en blanco re regres, de no ser así empuja el caracter actual en el buffer
        if ((caracterActual == ' ' || caracterActual == '\n' || caracterActual == '\t')) {
            return;
        }
        else {
            identifier.push_back(caracterActual);
        }

        while (!aceptarEstados[estado] && !errorEstados[estado]) {
            // Empieza con nuevo estado
            restartChar(); // Reinicia las vars booleanas del tipo de caracter
            checkCharacter(caracterActual);  //Checa si e caracter es letra,num,especial o espacio
            indicarChar(); // Obtiene characterType (Index de tabla de transiciones)
            int nuevoEstado = tablaTrans[estado][characterType];

            if (avanEstados[estado][characterType]) {
```

```
            if (avanEstados[estado][characterType]) {

                // getCharType regresa : 0 letra, 1 num, 2 especial, 3 espacio
                // Si es letra Y el caracter de enfrente no es un num Y el siguiente estado es aceptor
                if (getCharType(caracterActual) == 0 && !(getCharType(fin.peek()) != 1 && aceptarEstados[tablaTrans[nuevoEstado][peekCharType(fin.peek())]])) {
                        siguienteChar();
                        identifier.push_back(caracterActual);
                }
                // Si es un num Y el de enfrente no es letra Y el siguiente estado es aceptor
                else if (getCharType(caracterActual) == 1 && !(getCharType(fin.peek()) != 0 && aceptarEstados[tablaTrans[nuevoEstado][peekCharType(fin.peek())]])) {
                    siguienteChar();
                    identifier.push_back(caracterActual);
                }
                // Si los estados son comentarios sigue empujandolos al buffer
                else if (estado == 4 || estado == 5) {
                    siguienteChar();
                    identifier.push_back(caracterActual);
                }
                // peekCharType especifica el caracter especial
                // Si el caracter actual es Espacio en blanco o el caracter de enfrente es aceptor
                else if ( aceptarEstados[tablaTrans[nuevoEstado][peekCharType(fin.peek())]]) {
                    if (caracterActual == EOF) {
                        return;
                    }
                    else {
                        //identifier.push_back(characters);
                        nuevoEstado = tablaTrans[nuevoEstado][peekCharType(fin.peek())];
                    }
                }
                // Para espacios en blanco
                else if (getCharType(caracterActual) == 3) {
                    identifier = "";
                    siguienteChar();
                    identifier.push_back(caracterActual);
                }

```

```
            else {
                if (fin.peek() == EOF) {
                    mostrarToken(tablaTrans[nuevoEstado][peekCharType(fin.peek())]);
                    return;
                }
                else {
                    caracterActual = fin.peek();
                }
            }

        }
        //cout << "ESTADO: " << nuevoEstado << "\n";
        estado = nuevoEstado;
    }


    // Aqui recolecta e imprime el token
    if (aceptarEstados[estado]) {

        if ((caracterActual == '<' && fin.peek() == '=') || (caracterActual == '=' && fin.peek() == '=') || (caracterActual == '>' && fin.peek() == '=') || (caracterActual ==
            caracterActual = fin.peek();
            identifier.push_back(caracterActual);
            mostrarToken(estado);
            identifier = "";
            siguienteChar();
        }
        else {
            mostrarToken(estado);
            identifier = "";
        }
    }
    else {
        mostrarToken(estado);
    }

}
```

```
// Recorre estados hasta que se termina el archivo
void recorrerEstados() {
    while (fin.peek() != EOF) {
        estadoRecorrido();
    }
}

// Recorre estados y luego imprime el vectorOutput para obtener los resultados
int main()
{
    recorrerEstados();
    for (int x = 0; x < vecOutput.getActualPos();) {
        cout <<"<" + vecOutput.getIdent(x) + ">"<< endl;
        x++;
    }
}
```

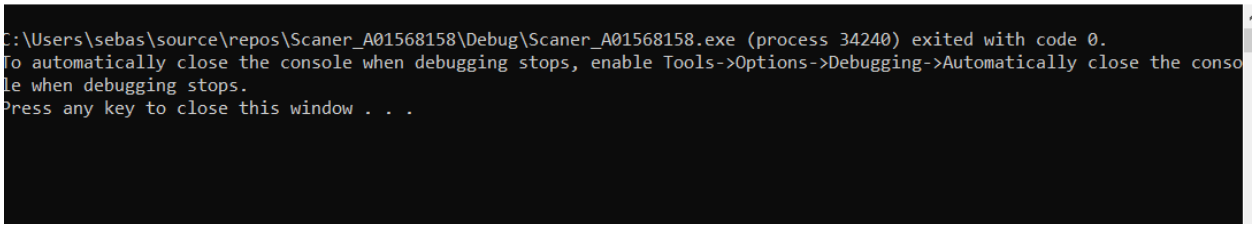
5. Verificación y Validación

5.1. Casos de Prueba

# Caso de Prueba	Descripción	Datos del Test	Resultado Esperado	Resultado Generado	Falló/Pasó
1	Recibir entrada de un comentario(/* */).	Entrada: “/*hola*/”	Salida vacía y término del programa.	Salida vacía y término del programa.	Pasó
2	Recibir de entrada un archivo vacío.	Entrada: vacío	Término del programa.	Término del programa.	Pasó
3	Recibir de entrada los caracteres compuestos.	Entrada: “<=”	<12>	<12>	Pasó
4	Recibir de entrada un comentario dentro de otro comentario.	Entrada: “/*hola/*como estas*/*/”	<23> <22>	<23> <22>	Pasó
5	Recibir de entrada diferentes combinaciones de caracteres.	Entrada: “a b c abc a b c”	<28,0> <28,1> <28,2> <28,3> <28,0> <28,1> <28,2>	<28,0> <28,1> <28,2> <28,3> <28,0> <28,1> <28,2>	Pasó
6	Recibir de entrada un caracter compuesto obligatorio pero solo la primera parte.	Entrada: “!”	-----ERROR, no se encontró '=' después de '!'-----	-----ERROR, no se encontró '=' después de '!'-----	Pasó

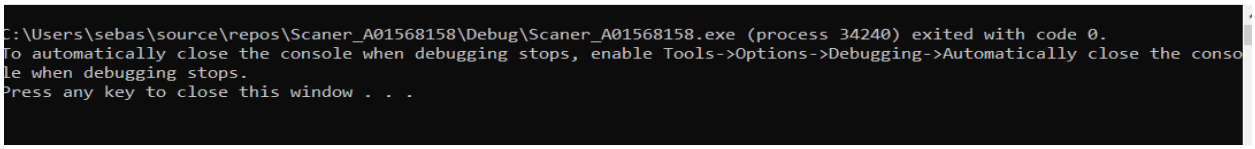
7	Recibir de entrada un caracter que no sea reconocido por el lenguaje.	Entrada: “ñ”	----ERROR, caracter no valido encontrado----	----ERROR, caracter no valido encontrado----	Pasó
8	Recibir de entrada una letra seguido de un número.	Entrada: “1a”	------ERROR, letra después de dígito------	------ERROR, letra después de dígito------	Pasó
9	Recibir de entrada un número seguido de una letra.	Entrada: “a1”	------ERROR, dígito después de letra-----	------ERROR, dígito después de letra-----	Pasó
10	Recibir de entrada caracteres compuestos juntos.	Entrada: “!===<=>=”	<15> <13> <12> <14>	<15> <13> <12> <14>	Pasó

Test 1



Se recibe de entrada un comentario el cual no regresa nada como resultado.

Test 2



Se recibe una entrada vacía la cual no regresa nada como resultado.

Test 3

```
<12>

C:\Users\sebas\source\repos\Scaner_A01568158\Debug\Scaner_A01568158.exe (process 36824) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the cons
le when debugging stops.
Press any key to close this window . . .
```

Se recibe una entrada con el caracter compuesto “<=” que regresa como resultado un token 12.

Test 4

```
<23>
<22>

C:\Users\sebas\source\repos\Scaner_A01568158\Debug\Scaner_A01568158.exe (process 34088) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Se recibe una entrada con un comentario dentro de otro comentario el cual regresa como resultado token 23 y token 22.

Test 5

```
<28,0>
<28,1>
<28,2>
<28,3>
<28,0>
<28,1>
<28,2>

C:\Users\sebas\source\repos\Scaner_A01568158\Debug\Scaner_A01568158.exe (process 10840) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Se recibe de entrada caracteres separados, una unión de estos en una cadena de caracteres sin espacios y luego los caracteres ya mencionados separados de nuevo. Regresa como resultado el token 28 de identificador con sus respectivos ids.

Test 6

```
 -----ERROR, no se encontr≤ '=' despu0s de '!'-----
C:\Users\sebas\source\repos\Scaner_A01568158\Debug\Scaner_A01568158.exe (process 13264) exited with code 3.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

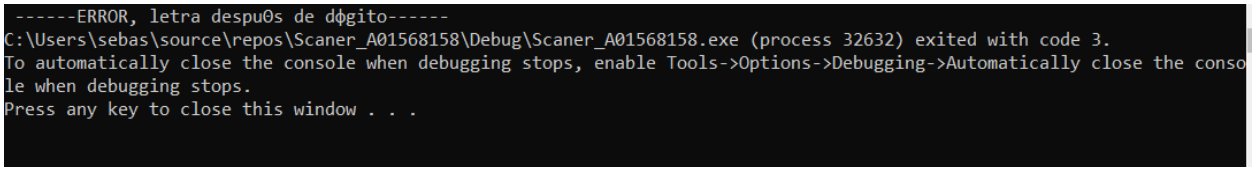
El caracter compuesto de ‘!=’, esta vez solo se recibió de entrada el caracter simple ‘!’ el cual no se identifica como un caracter simple y que arroja de salida un mensaje de error.

Test 7

```
 ----ERROR, caracter no valido encontrado----
C:\Users\sebas\source\repos\Scaner_A01568158\Debug\Scaner_A01568158.exe (process 34768) exited with code 3.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

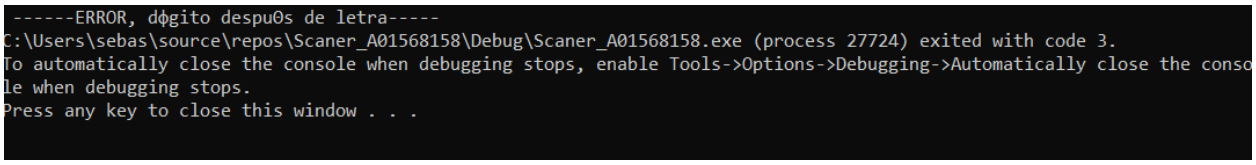
Se recibe de entrada un caracter no aceptado por el lenguaje correspondiendo a ‘ñ’, el cual arroja un mensaje de error como salida.

Test 8



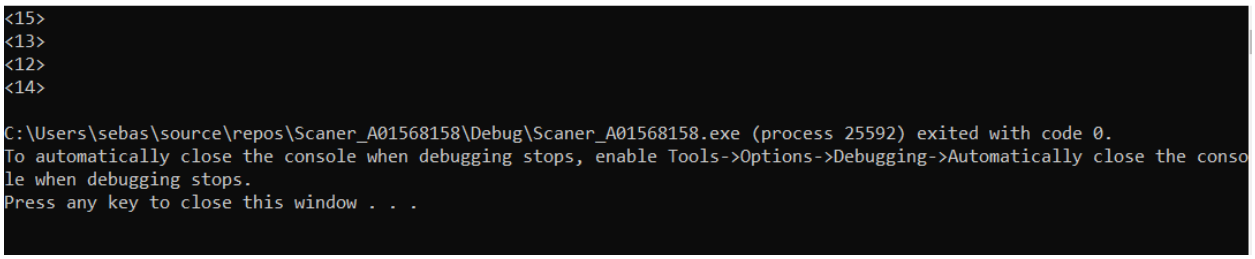
Se recibe como entrada un caracter después de un número, el cual arroja un error como salida.

Test 9



Se recibe como entrada un número después de un caracter, el cual arroja un error como salida.

Test 10



Se recibió una cadena de caracteres compuestos, la cual dio como resultado sus respectivos tokens separados correspondientes,15,13,12 y 14.

6. Referencias

“*What is Table-Driven Design?*”.Techopedia.com. 2022. 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. Accesado 29 de April 2022.

“*Finite state machines*”. isaaccomputerscience.org. 2022. https://isaaccomputerscience.org/concepts/dsa_toc_fsm?examBoard=all&stage=all Accesado 29 de Abril 2022.

R. Castelló, Lectura de Clase, Topic: “*Chapter 2 – Lexical Analysis.*” TC3048, Escuela de Ingeniería y Ciencias, ITESM, Chihuahua, Chih, Abril, 2022.

SAKSHAM894954,“*Tabla de Transición en Autómatas*”,acervolima.com. 2022. https://es.acervolima.com/tabla-de-transicion-en-automatas/
Accesado 29 de Abril 2022.

"*Diagrama de flujo*", concepto.de. 2022. https://concepto.de/diagrama-de-flujo/
Accesado 29 de Abril 2022.

Thomas Hamilton, "*How to Write Test Cases: Sample Template with Examples*", guru99.com. 2 de Abril , 2022. https://www.guru99.com/test-case.html
Accesado 29 de Abril 2022.

ELLIOT B. KOFFMAN,PAUL A. T. WOLFGANG ," *OBJECTS, ABSTRACTION, DATA STRUCTURES AND DESIGN USING C++*" ,John Wiley & Sons, Inc. ,29 de Abril 2022.