

# Desarrollo y Análisis de un Escáner 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

<b>1. Introducción.</b>	<b>3</b>
1.1. Resumen.	3
1.2. Notación.	3
<b>2. Análisis.</b>	<b>5</b>
2.1. Requerimientos	5
2.2. AFD	9
2.3. Tabla de Transiciones	10
2.4. Tabla de Símbolos.	10
2.5. Mensajes de Error	11
<b>3. Diseño.</b>	<b>11</b>
3.1. Pseudocódigo	11
3.2. Diagrama de Flujo	13
3.3. AFD	13
3.4. Lista de Tokens	15
3.5. Tabla de Símbolos	15
<b>4. Implementación</b>	<b>16</b>
4.1. Código Fuente	16
<b>5. Verificación y Validación.</b>	<b>25</b>
5.1. Casos de Prueba	25
<b>6. Referencias.</b>	<b>28</b>

# 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]**

**dígito = [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**

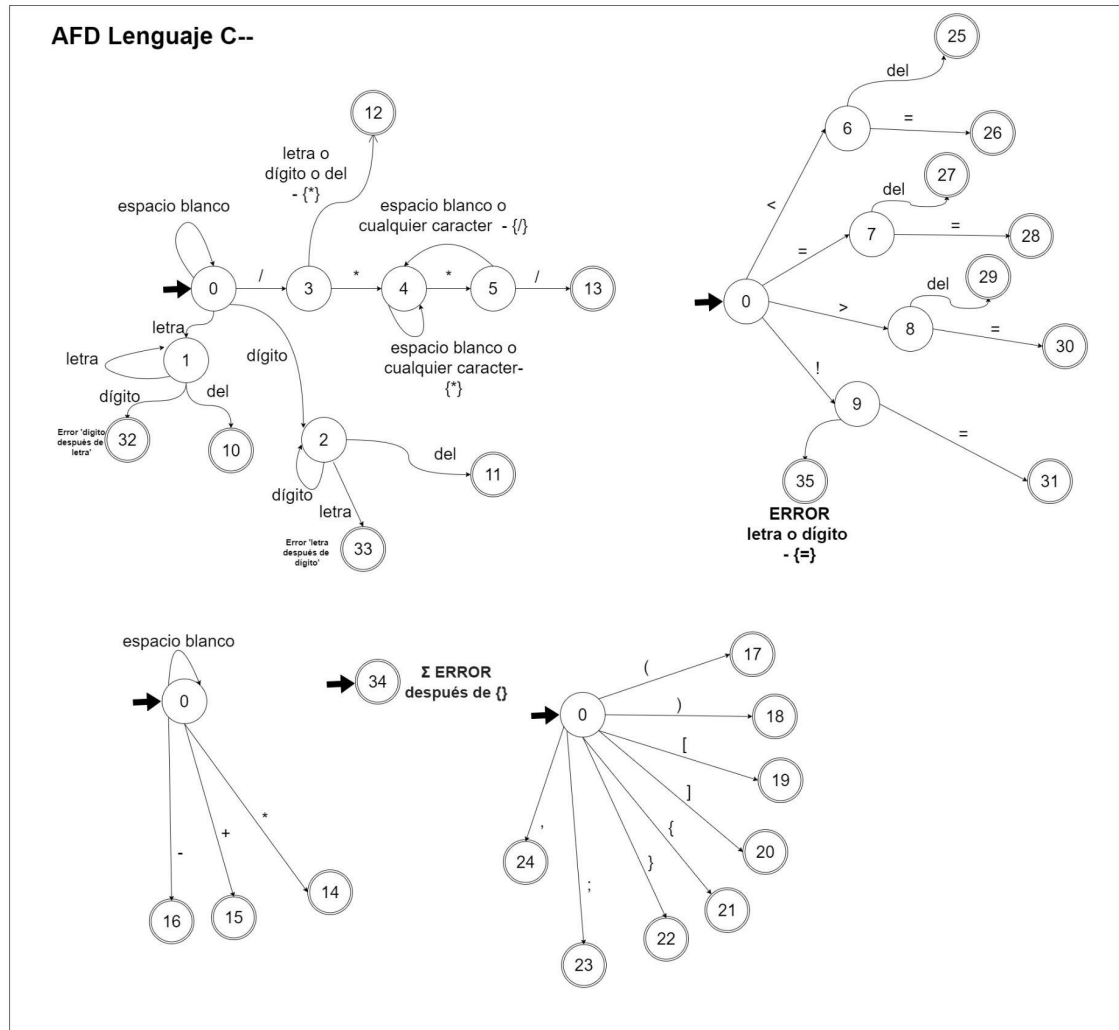
- Conjunto de delimitadores

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

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1		letra	dígito	/	*	+	-	<	=	>	!	(	)	[	]	{	}	;	,	Espacio Blanco	Caracter raro	
2	0	1	2	3	14	15	16	6	7	8	9	17	18	19	20	21	22	23	24	0	34	
3	1	33	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	34	
4	2	12	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	34	
5	3	12	12	12	4	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	34	
6	4	4	4	4	5	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	34	
7	5	4	4	13	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	34	
8	6	25	25	25	25	25	25	25	26	25	25	25	25	25	25	25	25	25	25	25	34	
9	7	27	27	27	27	27	27	27	28	27	27	27	27	27	27	27	27	27	27	27	34	
10	8	29	29	29	29	29	29	29	30	29	29	29	29	29	29	29	29	29	29	29	34	
11	9	35	35	35	35	35	35	35	35	35	35	35	35	35	35	35	35	35	35	35	34	
12	10																				Identificador	
13	11																				num	
14	12																				/	
15	13																				comentario	
16	14																				*	
17	15																				-	
18	16																				[	
19	17																				]	
20	18																				{	
21	19																				}	
22	20																				;	
23	21																				,	
24	22																					
25	23																					
26	24																					
27	25																					
28	26																					
29	27																					
30	28																					
31	29																					
32	30																					
33	31																					
34	32																				ERROR	
35	33																				ERROR	
36	34																				ERROR	
37	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
checharCaracter(){
    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
}

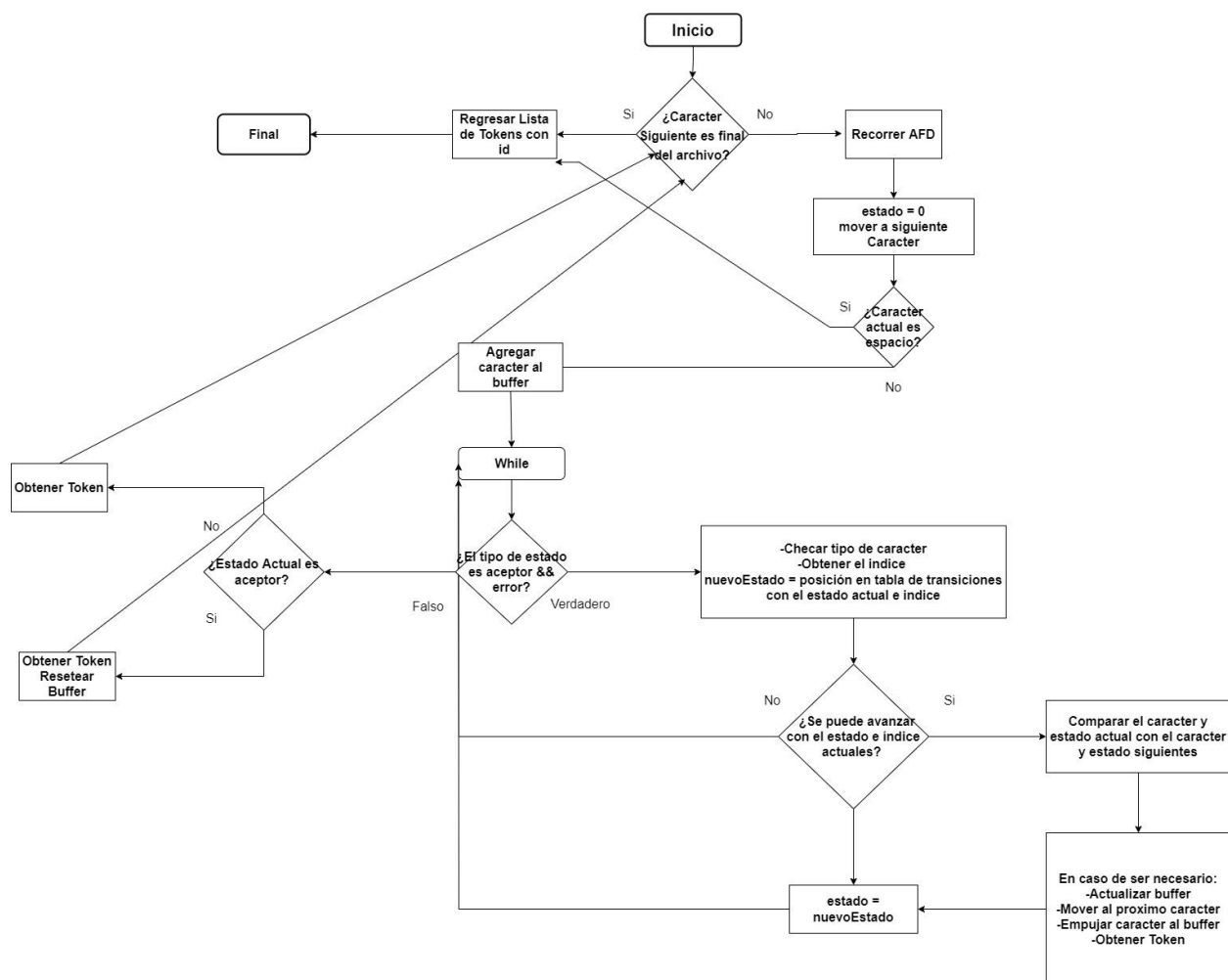
```

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. Diagrama 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, 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, 34},
{33, 2, 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, 34},
{4, 4, 4, 5, 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},
{25, 25, 25, 25, 25, 25, 25, 26, 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, 34},
{29, 29, 29, 29, 29, 29, 29, 30, 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, 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 carácter 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

```

1 // Scanner_AB1568158.cpp
2 // Table-Driven
3 // Hecho por Sebastian Salazar Villanueva
4 #include <iostream>
5 #include <fstream>
6 #include <string>
7 using namespace std;
8
9 // Variables Globales
10 fstream fin("Sample.txt", fstream::in); // Modo lectura File in
11 bool aceptarChar; // Llave para saber si un caracter es letra, numero o caracter especial, de esta forma no pasar por los demás
12 string identifier; // Buffer donde se guarda cada uno de los caracteres ya sean
13 int characterType; // Indica si es letra, numero o caracter especial
14
15 // El caracter actual que se lee y los Arreglos de Caracteres
16 char caracterActual = ' ';
17 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','W','X','Y','Z' };
18 int numArr[10] = { '0','1','2','3','4','5','6','7','8','9' };
19 char caracEspecialesArr[16] = { ' ','/','~','!','@','#','$','%','^','&','*','(',')','{','}','[','\'],'\"','\'' };
20 char espaciosArr[4] = { '\n','\t',' ','\0' };
21
22 // Lista de Tokens
23 string lista_token[27] = { "RETURN", "WHILE", "VOID", "ELSE", "IF", "INT", "INPUT", "OUTPUT", ">", "<", "=", "<=", ">=", "!=", "((", ")", "[", "]", "{", "}", "/", "++", "--", "+", "-", "." };
24
25 // Llave para saber que tipo de caracter es el caracter actual
26 bool esLetra = false;
27 bool esNum = false;
28 bool esEspecial = false;
29 bool esEspacio = false;
30
31 // Estados, se especifica cuales estados son aceptores, de error, un arreglo bidimensional para revisar si puede avanzar
32 bool aceptarEstados[36] = { 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, true, true, true };
33 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 };
34 bool avanzarEstados[10][20] = {
35     { true, true, true, false, false, false, true, true, true, true, false, false, false, false, false, false, false, false, false, false },
36     { true, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false },
37     { false, true, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false },
38     { false, false, false, true, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false },
39     { true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, true },
40     { false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false },
41     { false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false },
42     { false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false },
43     { false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false },
44     { false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false, false },
45 };

```

```

46 //Tabla de transiciones
47 int tablaTrans[10][20] = {
48     { 1, 2, 3, 14, 15, 16, 6, 7, 8, 9, 17, 18, 19, 20, 21, 22, 23, 24, 0, 34 },
49     { 1, 32, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 34 },
50     { 33, 2, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 34 },
51     { 12, 12, 12, 4, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 34 },
52     { 4, 4, 4, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4 },
53     { 4, 4, 11, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4 },
54     { 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 34 },
55     { 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 34 },
56     { 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 34 },
57     { 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 34 },
58 };
59
60 namespace Vec { ... }
61 // "Vectores con sus tipos de datos"
62 Vec::vector<string> vecOutput; // Vector para la salida
63 Vec::vector<int> vecNum; // Vector para los números
64 Vec::vector<string> vecLetra; // Vector para los identificadores
65
66 // Transforma todas las letras de un identificador para revisar si es una Keyword
67 string upperCase(string ident) {
68     string upper = "";
69     for (int i = 0; i < ident.length(); i++) {
70         upper += toupper(ident[i]);
71     }
72     return upper;
73 }
74
75 // Compara en que estado se encuentra y regresa Token
76 int compararToken(int estado) {
77     string upper = upperCase(identifier);
78     switch (estado) {
79     case 10:
80         for (int d = 0; d <= 7; d++) {
81             if (upper == lista_token[d]) {
82                 return d + 1;
83             }
84         }
85         d++;
86     }
87     return 28;
88 }
89 break;

```



```

61 namespace Vec {
62     template<typename Id>
63     class vector {
64     private:
65         // La capacidad inicial del arreglo con los 29 tokens
66         //size_t tipo de entero sin ser asignado
67         static const size_t capacidad_inicio = 1;
68         // La capacidad actual del arreglo
69         size_t capacidad_actual;
70         // El num_obj actual del arreglo
71         size_t num_obj;
72         //El array que contiene la data
73         Id* arr_datos;
74     public:
75         //Constructor
76         vector<Id>() : capacidad_actual(capacidad_inicio), arr_datos(new Id[capacidad_inicio]), num_obj(0) { }
77         // Regresa el numero de objetos que estan insertados, los cuales sirven para recorrer todo el vector de output
78         size_t getActualPos(){ ... }
79         // Obtiene lo que se encuentra en la posición especificada en el parámetro
80         Id& getIdent(int x) {
81             return arr_datos[x];
82         }
83         // Expande el vector para que sea posible insertar
84         void reservar(size_t capacidad_nueva){ ... }
85         // Insertar objeto en el vector especificado dependiendo de su tipo de dato
86         void insertar(size_t index, const Id& valor){ ... }
87     };
88
89     // *Vectores con sus tipos de datos*
90     Vec::vector<string> vecOutput; // Vector para la salida
91     Vec::vector<int> vecNum; // Vector para los números
92     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.

```

156 case 11:
157     return 29;
158     break;
159 case 12:
160     return 22;
161     break;
162 case 13:
163     return 0;
164     break;
165 case 14:
166     return 23;
167     break;
168 case 15:
169     return 25;
170     break;
171 case 16:
172     return 24;
173     break;
174 case 17:
175     return 16;
176     break;
177 case 18:
178     return 17;
179     break;
180 case 19:
181     return 18;
182     break;
183 case 20:
184     return 19;
185     break;
186 case 21:
187     return 20;
188     break;
189 case 22:
190     return 21;
191     break;
192 case 23:
193     return 27;
194     break;
195 case 24:
196     return 26;
197     break;

```

```

197         break;
198     case 25:
199         return 10;
200         break;
201     case 26:
202         return 12;
203         break;
204     case 27:
205         return 11;
206         break;
207     case 28:
208         return 13;
209         break;
210     case 29:
211         return 9;
212         break;
213     case 30:
214         return 14;
215         break;
216     case 31:
217         return 15;
218         break;
219     case 32:
220         cout << " -----ERROR, dígito después de letra----- ";
221         abort();
222         break;
223     case 33:
224         cout << " -----ERROR, letra después de dígito----- ";
225         abort();
226         break;
227     case 34:
228         cout << " ----ERROR, caracter no valido encontrado---- ";
229         abort();
230         break;
231     case 35:
232         cout << " -----ERROR, no se encontró '=' después de '!'-----";
233         abort();
234         break;
235     }
236 }
237

```

```

239 // Avanza a el proximo caracter a leer
240 void siguienteChar() {
241     fin >> noskipws >> caracterActual;
242 }
243
244 // Regresa el indice en la tabla de transiciones dependiendo de si es numero, letra, caracter especial o espacio
245 void indicarChar() {
246     if (esLetra == true) {
247         characterType = 0;
248     }
249     else if (esNum == true) {
250         characterType = 1;
251     }
252     else if (esEspecial == true) {
253         if (caracterActual == '/') {
254             characterType = 2;
255         }
256         else if (caracterActual == '*') {
257             characterType = 3;
258         }
259         else if (caracterActual == '+') {
260             characterType = 4;
261         }
262         else if (caracterActual == '-') {
263             characterType = 5;
264         }
265         else if (caracterActual == '<') {
266             characterType = 6;
267         }
268         else if (caracterActual == '=') {
269             characterType = 7;
270         }
271         else if (caracterActual == '>') {
272

```

```

279     else if (caracterActual == '>') {
280         characterType = 8;
281     }
282
283     else if (caracterActual == '!') {
284         characterType = 9;
285     }
286
287     else if (caracterActual == '(') {
288         characterType = 10;
289     }
290
291     else if (caracterActual == ')') {
292         characterType = 11;
293     }
294
295     else if (caracterActual == '[') {
296         characterType = 12;
297     }
298
299     else if (caracterActual == ']') {
300         characterType = 13;
301     }
302
303     else if (caracterActual == '{') {
304         characterType = 14;
305     }
306
307     else if (caracterActual == '}') {
308         characterType = 15;
309     }
310
311     else if (caracterActual == ';') {
312         characterType = 16;
313     }
314
315     else if (caracterActual == ',') {
316         characterType = 17;
317     }
318 }
319 else if (esEspacio == true) {

```

```

320     characterType = 18;
321 }
322 else {
323     characterType = 19;
324 }
325 }
326 }
327
328 // Checa si es una letra
329 void checkLetra(char characters) {
330     for (int i = 0; i <= 51; i++) {
331         if (characters == letraArr[i]) {
332             esLetra = true;
333             aceptchar = true;
334             i = 51;
335         }
336     }
337 }
338
339 // Checa si es un Numero
340 void checkNum(char characters) {
341     if (aceptchar == false) {
342         for (int x = 0; x <= 9; x++) {
343             if (characters == numArr[x]) {
344                 aceptchar = true;
345                 esNum = true;
346                 x = 9;
347             }
348         }
349     }
350 }
351
352 // Checa si es un caracter especial
353 void checkSpecial(char characters) {
354     if (aceptchar == false) {
355         for (int y = 0; y <= 15; y++) {
356             if (characters == caracEspecialesArr[y]) {
357                 esEspecial = true;
358                 aceptchar = true;
359             }
360         }
361     }
362 }

```

```

361         y = 17;
362     }
363     y++;
364 }
365 }
366 }
367
368 // Checa si es un espacio
369 void checkEspacios(char characters) {
370     if (acceptchar == false) {
371         for (int y = 0; y <= 4; y++) {
372             if (characters == espaciosArr[y]) {
373                 esEspacio = true;
374                 acceptchar = true;
375                 y = 3;
376             }
377             y++;
378         }
379     }
380 }
381
382 // Se encarga de checar que tipo de caracter es
383 void checkCharacter(char characters) {
384     if (acceptchar == false) {
385         checkLetra(characters);
386     }
387     if (acceptchar == false) {
388         checkNum(characters);
389     }
390
391     if (acceptchar == false) {
392         checkSpecial(characters);
393     }
394
395     if (acceptchar == false) {
396         checkEspacios(characters);
397     }
398 }
399
400 // Regresa que tipo de caracter es para luego ser usado como index en peekCharType
401 int getCharType(char characters) {

```

```

399
400 // Regresa que tipo de caracter es para luego ser usado como index en peekCharType
401 int getCharType(char characters) {
402
403     if (characters == EOF) {
404         return 3;
405     }
406     else {
407         for (int i = 0; i <= 51; i++) {
408             if (characters == letraArr[i]) {
409                 return 0;
410             }
411             i++;
412         }
413         for (int x = 0; x <= 9; x++) {
414             if (characters == numArr[x]) {
415                 return 1;
416             }
417             x++;
418         }
419
420         for (int y = 0; y <= 15; y++) {
421             if (characters == caracEspecialesArr[y]) {
422                 return 2;
423             }
424             y++;
425         }
426
427         for (int y = 0; y <= 5; y++) {
428             if (characters == espaciosArr[y]) {
429                 return 3;
430             }
431             y++;
432         }
433     }
434 }
435

```

```
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;
477         }
478     }
479 }
```

```

479     else if (c == '(') {
480         return 10;
481     }
482
483     else if (c == ')') {
484         return 11;
485     }
486
487     else if (c == '[') {
488         return 12;
489     }
490
491     else if (c == ']') {
492         return 13;
493     }
494
495     else if (c == '{') {
496         return 14;
497     }
498
499     else if (c == '}') {
500         return 15;
501     }
502
503     else if (c == ';') {
504         return 16;
505     }
506
507     else if (c == ',') {
508         return 17;
509     }
510 }
511 else if (getCharType(c) == 3) {
512     return 18;
513 }
514 else {
515     return 19;
516 }
517 }
518 }
519 }

```

```

519 // Resetea las llaves de el caracter
520 void restartChar() {
521     esLetra = false;
522     esNum = false;
523     esEspecial = false;
524     esEspacio = false;
525     aceptchar = false;
526 }
527
528 // Inserta el token, con su id(De ser necesario) en la el vector de output
529 void mostrarToken(int estado) {
530     if (compararToken(estado) == 0) {
531         //Ignorar comentario
532     }
533     else if (compararToken(estado) > 0 && compararToken(estado) < 28) {
534         vecOutput.insertar(vecOutput.getActualPos(), to_string(compararToken(estado)));
535     }
536     else if (compararToken(estado) == 28) {
537         bool existe = false;
538         int x;
539         for (x = 0; x < vecLetra.getActualPos(); ) {
540             if (identifier == vecLetra.getIdent(x)) {
541                 existe = true;
542                 break;
543             }
544             x++;
545         }
546         if (existe) {
547             vecOutput.insertar(vecOutput.getActualPos(), to_string(compararToken(estado)) + ',' + to_string(x));
548         }
549         else {
550             int index = vecLetra.getActualPos();
551             vecLetra.insertar(vecNum.getActualPos(), identifier);
552             vecOutput.insertar(vecOutput.getActualPos(), to_string(compararToken(estado)) + ',' + to_string(index));
553         }
554     }
555     else if (compararToken(estado) == 29) {
556         bool existe = false;
557         int x;
558     }
559 }

```

```

558     bool existe = false;
559     int x;
560     for (x = 0; x < vecNum.getActualPos(); ) {
561         if (identifier == to_string(vecNum.getIdent(x))) {
562             existe = true;
563             break;
564         }
565         x++;
566     }
567     if (existe) {
568         vecOutput.insertar(vecOutput.getActualPos(), to_string(compararToken(estado)) + ',' + to_string(x));
569     }
570     else {
571         int index = vecNum.getActualPos();
572         vecNum.insertar(vecNum.getActualPos(), stoi(identifier));
573         vecOutput.insertar(vecOutput.getActualPos(), to_string(compararToken(estado)) + ',' + to_string(index));
574     }
575 }
576 }
577
578 // Algoritmo del DFA, el cual facilita el manejo del caracter actual
579 void estadoRe corrido() {
580     int estado = 0;
581     siguienteChar();
582
583     // Si el valor es un espacio en blanco re regres, de no ser así empuja el caracter actual en el buffer
584     if ((caracterActual == ' ' || caracterActual == '\n' || caracterActual == '\t')) {
585         return;
586     }
587     else {
588         identifier.push_back(caracterActual);
589     }
590
591     while (!aceptarEstados[estado] && !errorEstados[estado]) {
592         // Empezar con nuevo estado
593         restartChar(); // Reinicia las vars booleanas del tipo de caracter
594         checkCharacter(caracterActual); // Checa si e caracter es letra,num,especial o espacio
595         indicarChar(); // Obtiene characterType (Index de tabla de transiciones)
596         int nuevoEstado = tablaTrans[estado][characterType];
597
598         if (avanEstados[estado][characterType]) {
599
600             // getCharType regresa : 0 letra, 1 num, 2 especial, 3 espacio
601             // Si es letra Y el caracter de enfrente no es un num Y el siguiente estado es aceptor
602             if (getCharType(caracterActual) == 0 && !(getCharType(fin.peek()) != 1 && aceptarEstados[nuevoEstado][peekCharType(fin.peek())])) {
603                 siguienteChar();
604                 identifier.push_back(caracterActual);
605             }
606             // Si es un num Y el de enfrente no es letra Y el siguiente estado es aceptor
607             else if (getCharType(caracterActual) == 1 && !(getCharType(fin.peek()) != 0 && aceptarEstados[nuevoEstado][peekCharType(fin.peek())])) {
608                 siguienteChar();
609                 identifier.push_back(caracterActual);
610             }
611             // Si los estados son comentarios sigue empujandolos al buffer
612             else if (estado == 4 || estado == 5) {
613                 siguienteChar();
614                 identifier.push_back(caracterActual);
615             }
616             // peekCharType especifica el caracter especial
617             // Si el caracter actual es Espacio en blanco o el caracter de enfrente es aceptor
618             else if (aceptarEstados[nuevoEstado][peekCharType(fin.peek())]) {
619                 if (caracterActual == EOF) {
620                     return;
621                 }
622                 else {
623                     //identifier.push_back(characters);
624                     nuevoEstado = tablaTrans[nuevoEstado][peekCharType(fin.peek())];
625                 }
626             }
627             // Para espacios en blanco
628             else if (getCharType(caracterActual) == 3) {
629                 identifier = "";
630                 siguienteChar();
631                 identifier.push_back(caracterActual);
632             }
633

```

```

634         else {
635             if (fin.peek() == EOF) {
636                 mostrarToken(tablaTrans[nuevoEstado][peekCharType(fin.peek())]);
637                 return;
638             }
639             else {
640                 caracterActual = fin.peek();
641             }
642         }
643     }
644 }
645 //cout << "ESTADO: " << nuevoEstado << "\n";
646 estado = nuevoEstado;
647 }
648
649
650 // Aqui recolecta e imprime el token
651 if (aceptarEstados[estado]) {
652     if ((caracterActual == '<' && fin.peek() == '=') || (caracterActual == '-' && fin.peek() == '-') || (caracterActual == '>' && fin.peek() == '=') || (caracterActual ==
653         caracterActual = fin.peek();
654         identifi er .push_back(caracterActual);
655         mostrarToken(estad o);
656         identifi er = "";
657         siguienteChar();
658     }
659     else {
660         mostrarToken(estad o);
661         identifi er = "";
662     }
663 }
664
665 else {
666     mostrarToken(estad o);
667 }
668
669 }
670

```

```

670
671 // Recorre estados hasta que se termina el archivo
672 void recorrerEstados() {
673     while (fin.peek() != EOF) {
674         estadoRecorrido();
675     }
676 }
677
678 // Recorre estados y luego imprime el vectorOutput para obtener los resultados
679 int main()
680 {
681     recorrerEstados();
682     for (int x = 0; x < vecOutput.getActualPos();) {
683         cout <<"<" + vecOutput.getIdent(x) + ">"<< endl;
684         x++;
685     }
686 }

```



## 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

```
C:\Users\sebas\source\repos\Scanner_A01568158\Debug\Scanner_A01568158.exe (process 34240) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

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

## Test 2

```
C:\Users\sebas\source\repos\Scanner_A01568158\Debug\Scanner_A01568158.exe (process 34240) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

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

### Test 3

```
<12>
C:\Users\sebas\source\repos\Scanner_A01568158\Debug\Scanner_A01568158.exe (process 36824) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console 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\Scanner_A01568158\Debug\Scanner_A01568158.exe (process 34088) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console 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\Scanner_A01568158\Debug\Scanner_A01568158.exe (process 10840) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console 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 encontró '=' después de '!'-----
C:\Users\sebas\source\repos\Scanner_A01568158\Debug\Scanner_A01568158.exe (process 13264) exited with code 3.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console 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\Scanner_A01568158\Debug\Scanner_A01568158.exe (process 34768) exited with code 3.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console 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

```
-----ERROR, letra después de dígito-----
C:\Users\sebas\source\repos\Scanner_A01568158\Debug\Scanner_A01568158.exe (process 32632) exited with code 3.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

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

### Test 9

```
-----ERROR, dígito después de letra-----
C:\Users\sebas\source\repos\Scanner_A01568158\Debug\Scanner_A01568158.exe (process 27724) exited with code 3.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

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

### Test 10

```
<15>
<13>
<12>
<14>

C:\Users\sebas\source\repos\Scanner_A01568158\Debug\Scanner_A01568158.exe (process 25592) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

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”. isaacomputerscience.org. 2022.

[https://isaacomputerscience.org/concepts/dsa\\_toc\\_fsm?examBoard=all&stage=all](https://isaacomputerscience.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.