

Universidad Rafael Landívar  
Facultad de Ingeniería  
Lenguajes Formales y Autómatas  
Ing. Moises Alonso



# **PROYECTO LENGUAJES FORMALES Y**

## **AUTÓMATAS**

### **FASE 1**

Integrantes: Megan Naómi Morales Betancourt 1221120  
Emilio Antonio Barillas Contreras 1150620  
Roberto Alfredo Moya Noack - 1273020  
Fecha de entrega: 27 de febrero del 2023

# INTRODUCCIÓN

El objetivo de este proyecto es construir un analizador léxico capaz de tomar un texto de entrada y separarlo en una secuencia de tokens o símbolos léxicos, que conforman los elementos básicos del lenguaje de programación que se desea analizar.

El desarrollo de un analizador léxico es una tarea fundamental en el proceso de construcción de un compilador completo. Es la primera etapa en la cadena de herramientas que convierte el código fuente en un programa ejecutable. Por lo tanto, su correcta implementación y funcionamiento es crucial para la generación de código correcto y eficiente.

Este proyecto se enfoca en la implementación de un analizador léxico para una gramática dada utilizando herramientas como expresiones regulares y árboles de expresiones.

# ANÁLISIS

Para la resolución de esta fase del proyecto se planteó recorrer el archivo guardando cada línea en una lista simple. Cada una de las partes de la lista nos serán de utilidad para ir analizando línea por línea el archivo.

El programa consiste en recorrer la lista en un ciclo, identificando cada una de las líneas por separado realizando distintos procedimientos para poder determinar si la entrada es correcta o no.

Para ello se utilizó una librería la cual nos permite utilizar expresiones regulares las cuales son de mucha utilidad para verificar las entradas de las líneas del archivo.

Para cada parte del archivo (SETS, TOKENS, ACTIONS y ERROR) se utilizaron procedimientos diferentes y expresiones diferentes:

## - SETS:

Para evaluar la sección de sets, se guardó el archivo txt en diferentes líneas de texto. Estas líneas fueron leídas y evaluadas para la palabra reservada SETS, en caso de no encontrarse se continuaba con los TOKENS.

De ser hallada la palabra SETS se procede a leer si estos constan del identificador LETRA, DIGITO o CHAR.

```
switch (id)
{
    case "\tLETRA":
        checkLETRA.checkLetra(rule, b);
        break;
    case "\tDIGITO":
        checkDIGITO.checkDigito(rule, b);
        break;
    case "\tCHARSET":
        checkCHARSET.checkCharset(rule, b);
        break;
}
```

Imagen No.1 – Switch case SETS

Para evaluar el identificador LETRA se utilizó la siguiente expresión regular, esta toma en cuenta las comillas, una letra y una separación de '..' o '+'.  
"(( )\*['][A-Z]|[a-z]|[\_][']([..]|[+]))+"))

Imagen No.2 – Expresión regular LETRA

$(( )^*[A-Z][a-z][\_]'([..][+]))^+$

```
graph TD; N1((+)) --> N2((|)); N1 --> N3(([+])); N2 --> N4((.)); N2 --> N5(([..])); N4 --> N6((|)); N4 --> N7(([])); N6 --> N8((|)); N6 --> N9(([])); N8 --> N10((|)); N8 --> N11(([a-z])); N10 --> N12((*)); N10 --> N13(([A-Z])); N12 --> N14(([])); N12 --> N15(( ));
```

Imagen No.3 – Árbol de expresión SETS LETRA

Para evaluar DIGITO se usa la siguiente expresión regular, la cual evalúa si hay un dígito entre comillas seguido o no de otro con '.' o '+' de por medio.

```
if (Regex.IsMatch(line, "(( )*'[0-9]'([..]|([+]))+)|"))  
{
```

Imagen No.4 – Expresión regular DIGITO

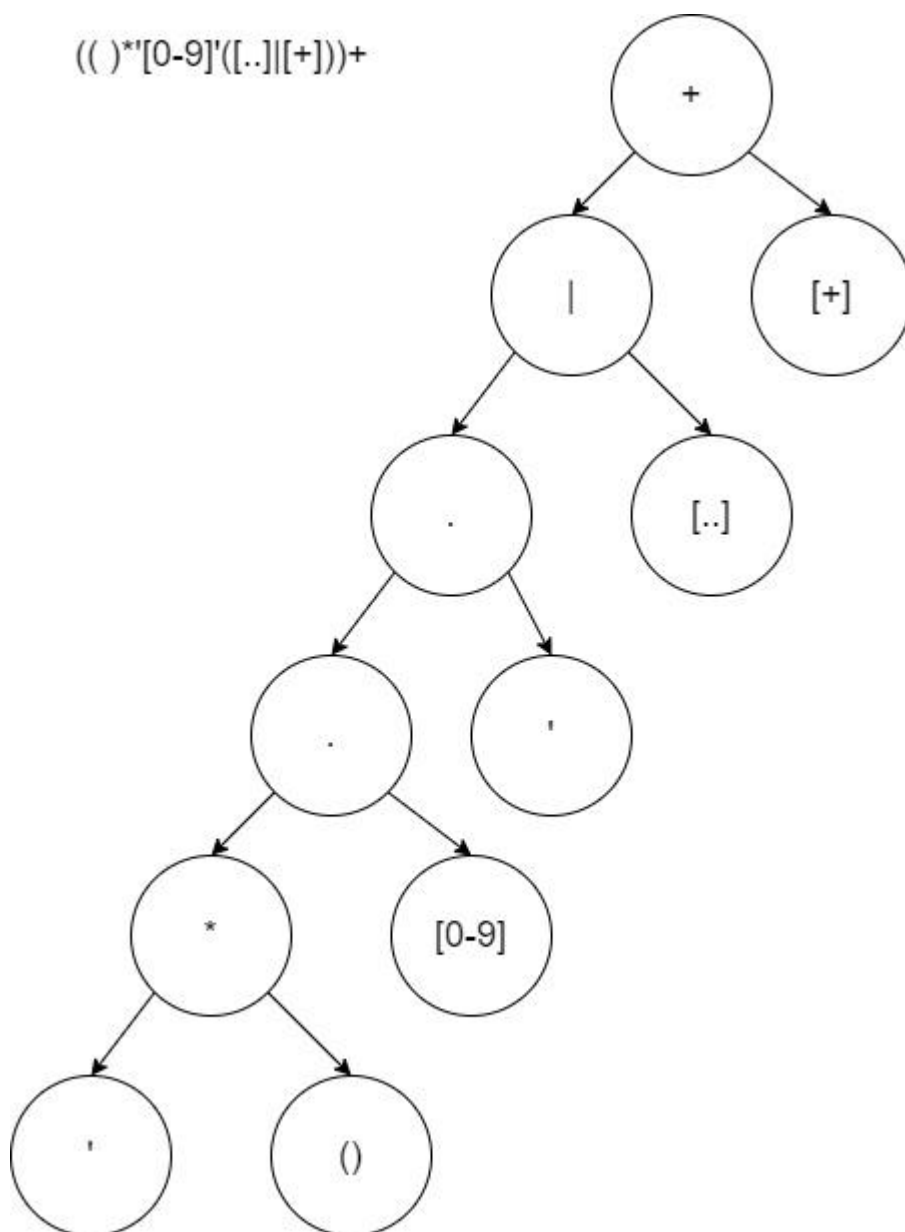


Imagen No.5 – Árbol de expresión SETS DIGITO

Finalmente el identificador CHAR se evalúa con la siguiente expresión que verifica que esté la palabra reservada CHR junto con un dígito de hasta 3 cifras.

```
if (Regex.IsMatch(line, "( )*CHR[ ]([0-9]{1,3})[ ]([. ]| [+]))+"))  
{
```

Imagen No.6 – Expresión regular CHAR



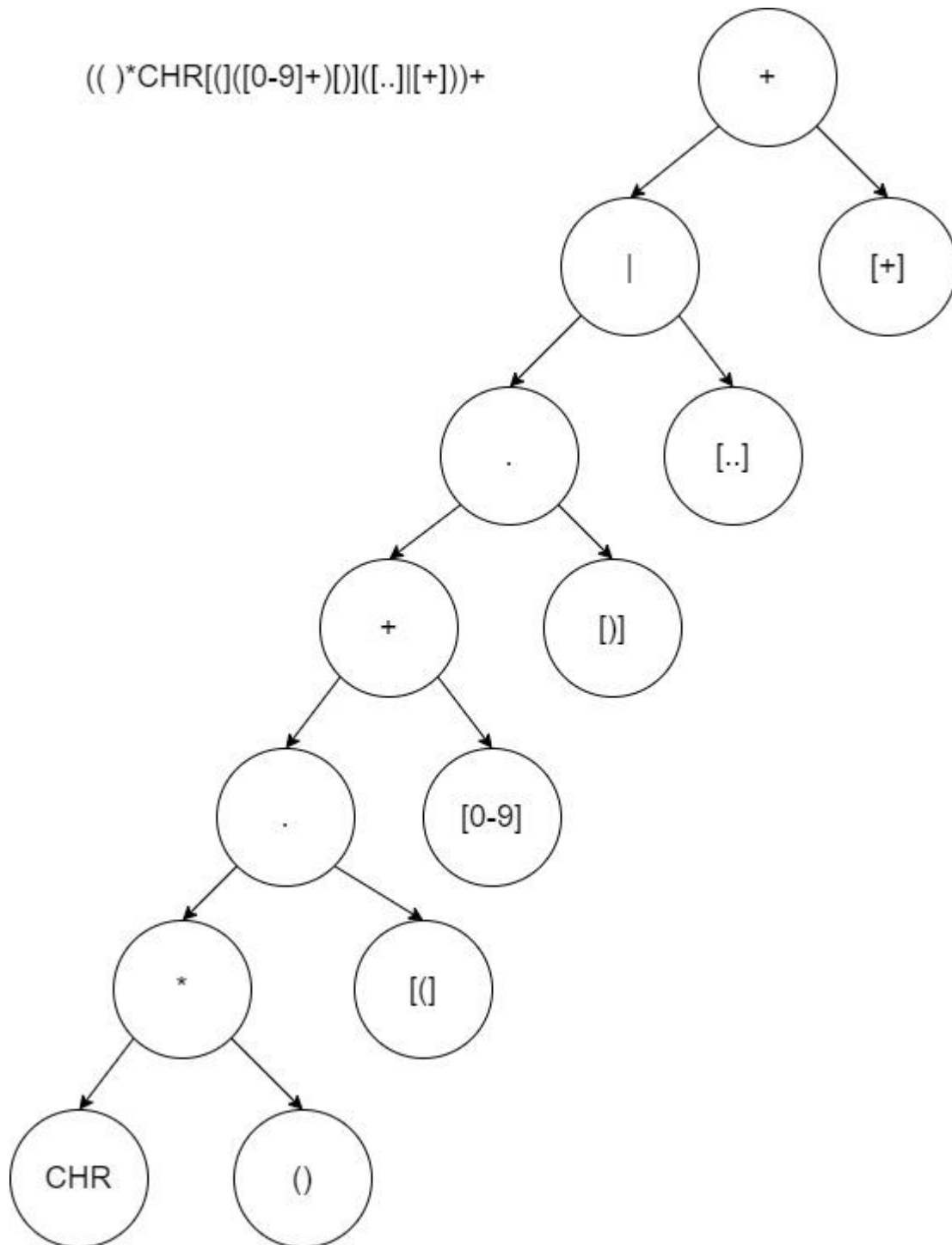


Imagen No.7 – Árbol de expresión SETS CHAR

#### - TOKENS:

Cuando el programa encuentre la palabra “TOKENS” entrará a un loop el cual se detendrá cuando encuentre la palabra “ACTIONS”. Verifica la expresión regular dependiendo del archivo de entrada.

Expresión regular creada:

$^((TOKEN(\\t)*[0-9]([0-9])*(\\t)*=(\\t)*)(\\t)*([\\' -~\\x80-\\xFF])*|([A-Z])*(\\t)*|(\\w+|\\w*\\w\\w\\w\\w([A-Z]|(\\t)|(\\w+|\\w*\\w\\w\\w\\w)*\\w\\w\\w\\w)*))\\$"$

Árbol de expresiones

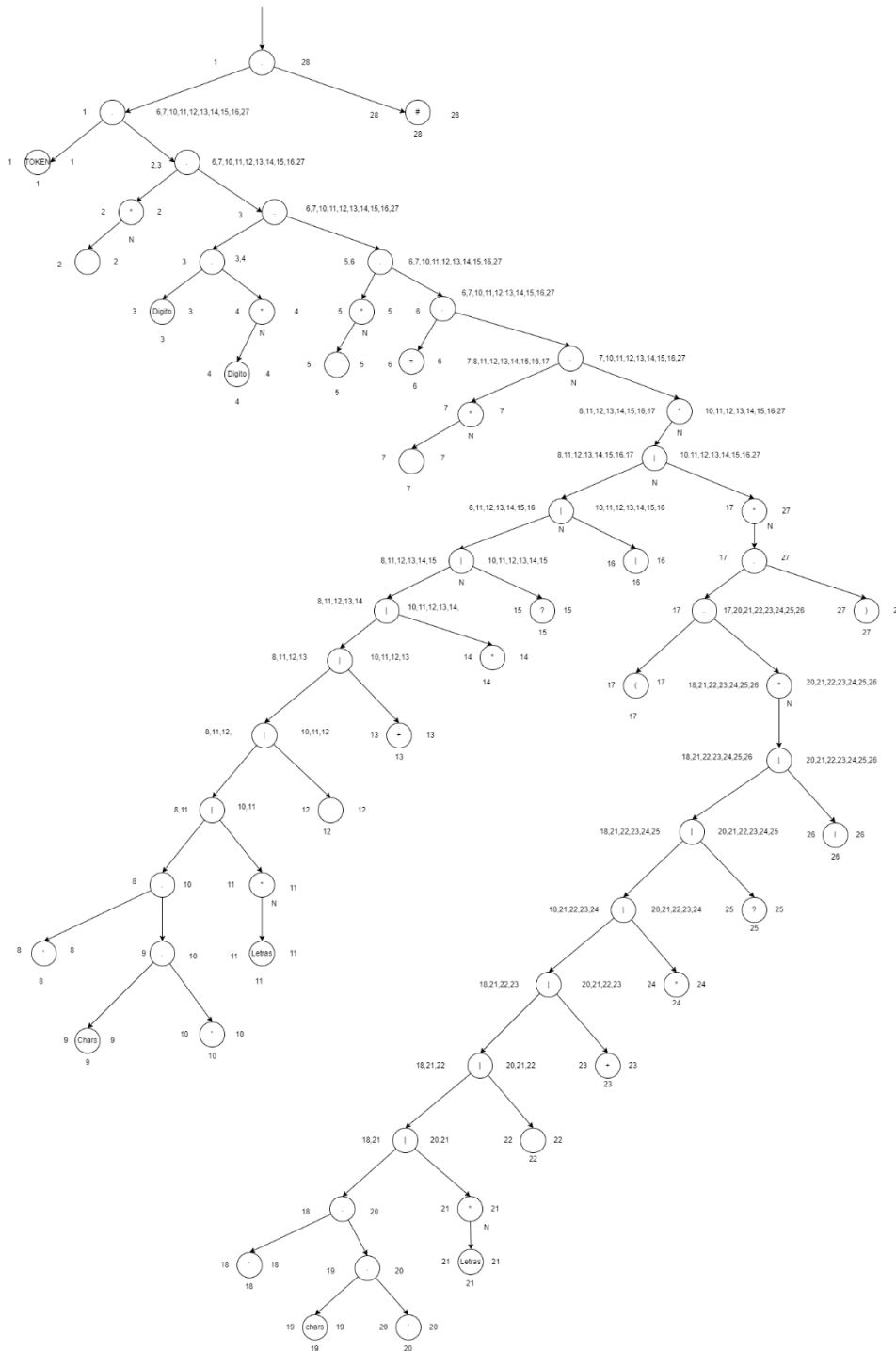


Imagen No.8 – Árbol de expresión TOKENS

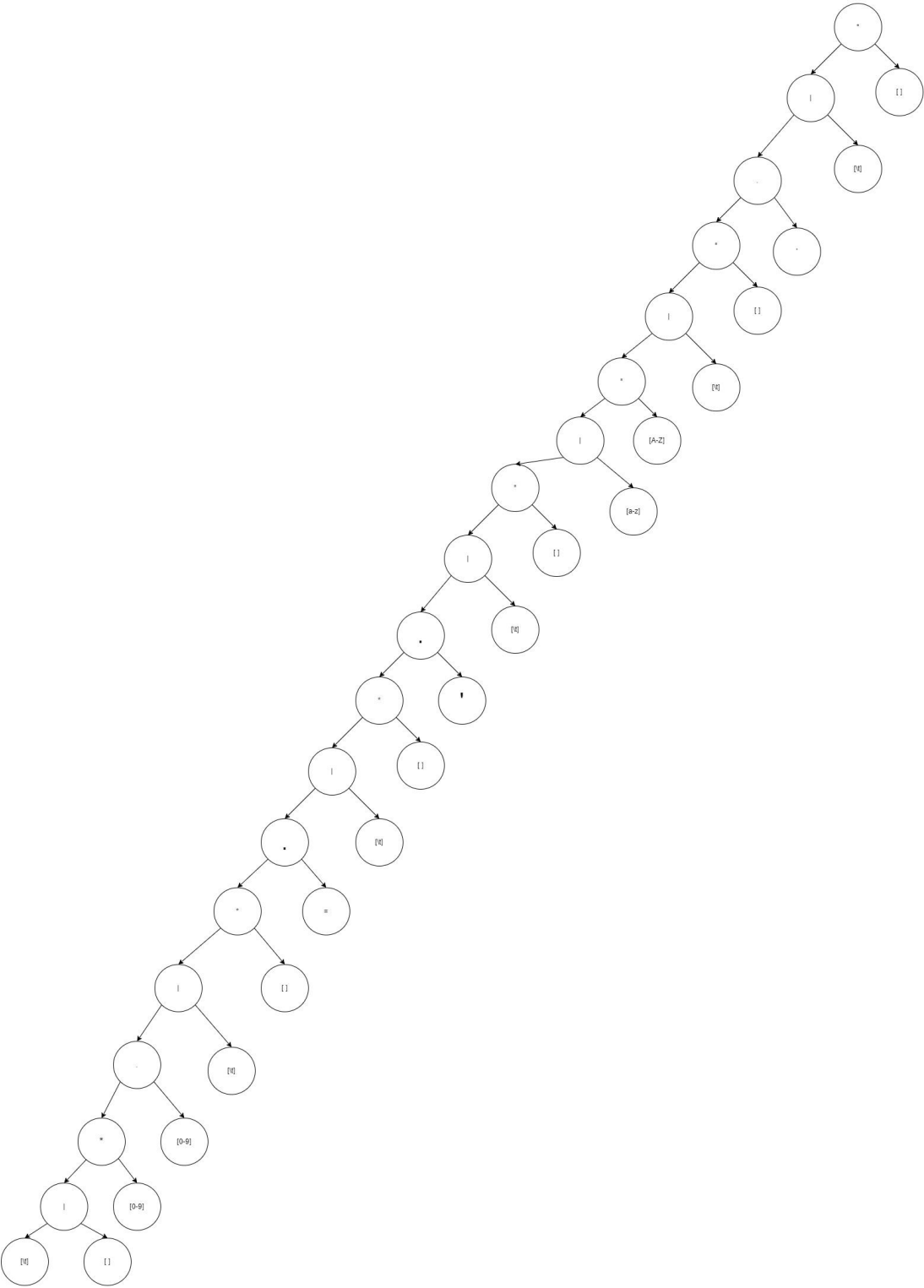
- **ACTIONS:**

Cuando el programa encuentre la palabra "ACTIONS" entrará a otro loop el cual recorrerá el mismo archivo buscando la palabra "Reservadas()" esto debido a que sí o sí debe venir luego de encontrar la palabra "ACTIONS". Seguido de esto verificará que se encuentre el abrir llaves. Luego entrará a otro ciclo y recorrerá el archivo verificando con la siguiente expresión regular:

```
"(((\\t|[ ])*[0-9][0-9])(\\t|[ ])*=(\\t|[ ])*'(\\t|[ ])*([a-z]|[A-Z])*(\\t|[ ])*'(\\t|[ ])*$"
```

Árbol de expresión:

$((\langle \Gamma \rangle [\langle \Gamma \rangle ])^* [0-9] (\langle \Gamma \rangle [\langle \Gamma \rangle ])^* = (\langle \Gamma \rangle [\langle \Gamma \rangle ])^* (\langle \Gamma \rangle [\langle \Gamma \rangle ])^* ([a-z] [\langle \Gamma \rangle ])^* (\langle \Gamma \rangle [\langle \Gamma \rangle ])^* )^*$



## Imagen No.9 – Árbol de expresión ACTIONS

Esta expresión me indica las acciones que pueden venir.

Al terminar busco una llave de cierre que me indique que salió. Cuando esto suceda verificaré con la siguiente expresión regular si se encontró otra función:

“(([\t]|[ ])\*[A-Z]\*([\t]|[ ])\*[()])([\t]|[ ])\*\$”

Árbol de expresión:

$(([t]|[ ])^*[A-Z]^*([t]|[ ])^*[()])([t]|[ ])^*\$$

```
graph TD; Root((*)) --> I1((I)); Root --> L1([]); I1 --> Dot((.)); I1 --> L2([]); Dot --> Star1((*)); Dot --> L3(()); Star1 --> I2((I)); Star1 --> L4([]); I2 --> Star2((*)); I2 --> L5([t]); Star2 --> Star3((*)); Star2 --> L6([A-Z]); Star3 --> I3((I)); Star3 --> L7([]); I3 --> L8([]); I3 --> L9([t]); L8 --> L10([]); L8 --> L11([t]);
```

Imagen No.10 – Árbol de expresión FUNCIONES

Esta expresión me indica si se encontró otra función. Una vez terminado esto pasará a los errores.

- **ERROR:**

Seguido de los ACTIONS, estará la siguiente expresión regular la cual verificará si se encuentra un error:

" ([\t][ ])\*ERROR([\t][ ])\*=[\t][ ]\*[0-9][0-9]( [\t][ ])\*"

Arbol de expresión:

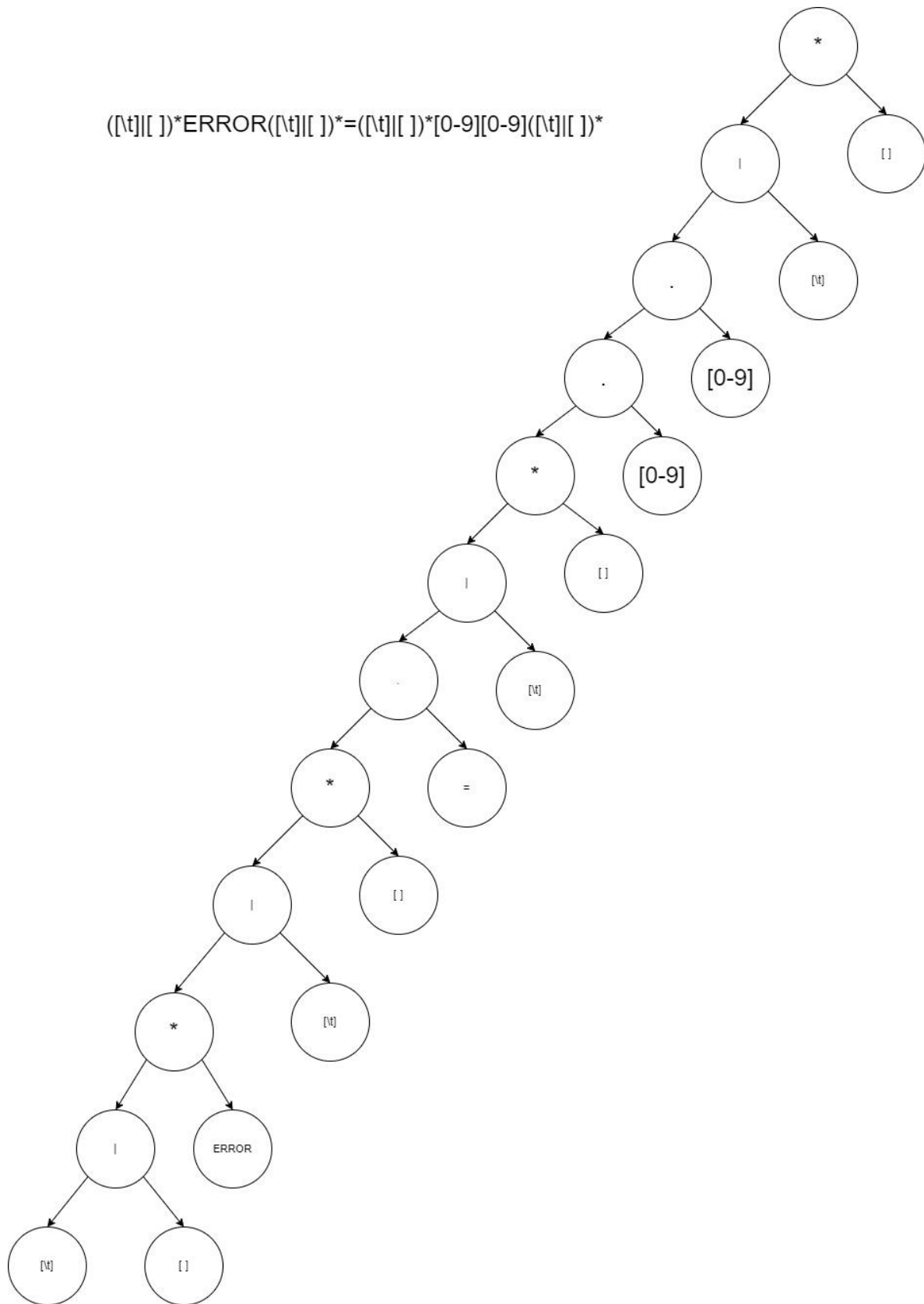


Imagen No.11 – Árbol de expresión ERROR





# DIAGRAMAS

Diagrama de flujo:

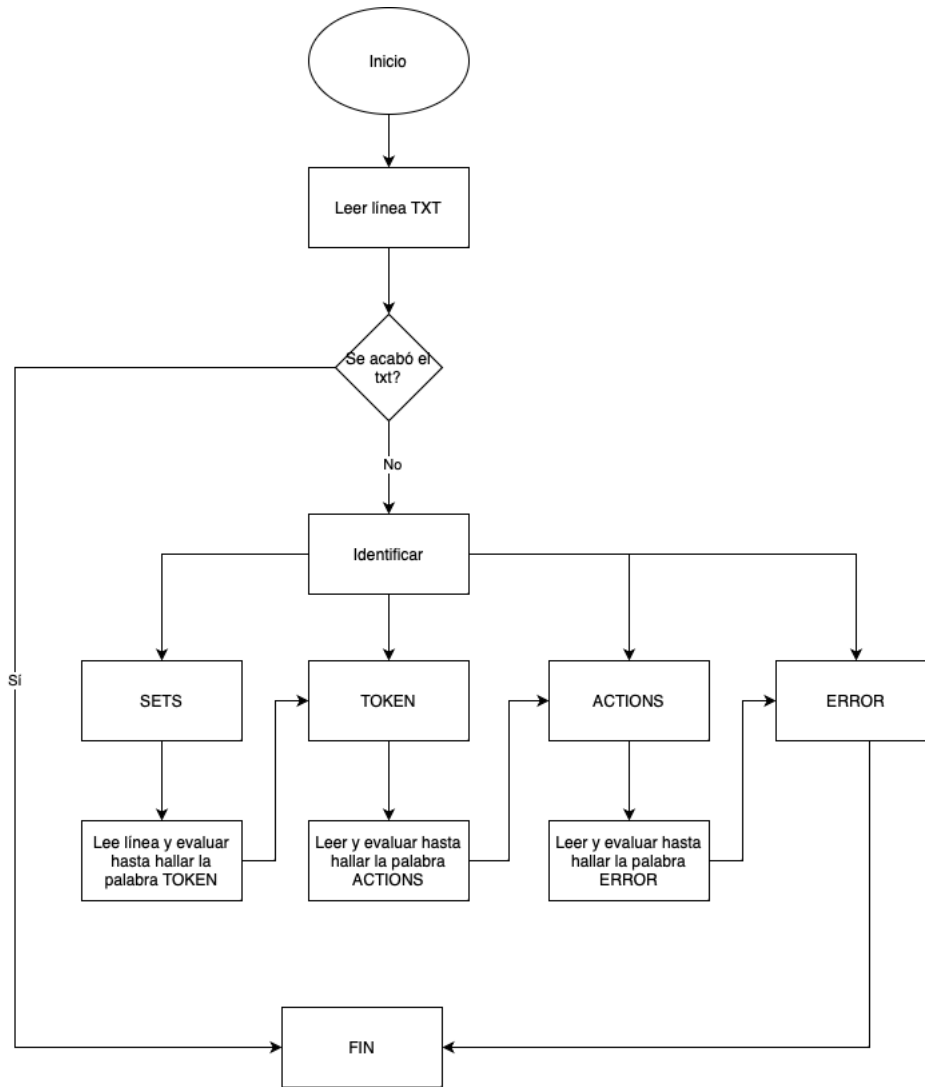


Imagen No.12 – Diagrama de flujo

## **CONCLUSIONES**

- Las expresiones regulares asisten al usuario al momento de escribir en cualquier gramática. Se aseguran de que el texto ingresado cumpla un formato específico para evitar errores de sintaxis.
- La gramática permite establecer las reglas con las que se escribe y rige cualquier lenguaje.
- A través de estructuras de datos es posible almacenar y evaluar las gramáticas para su uso al momento de evaluar textos ingresados por el usuario.

## ANEXOS

- Manual de usuario:

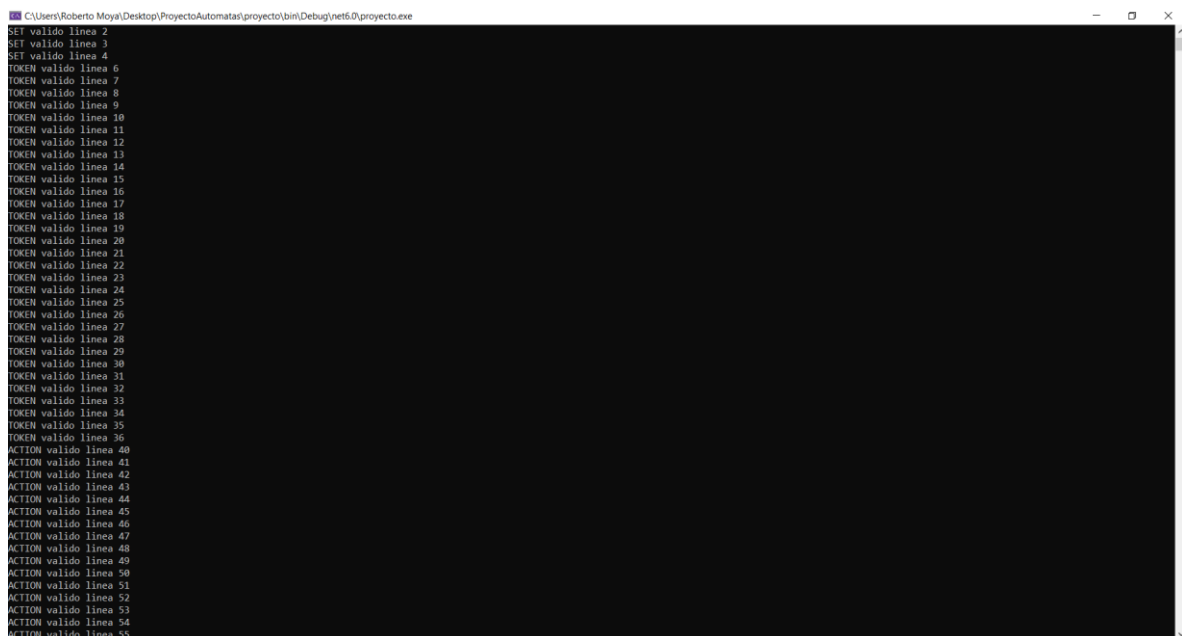
Para el correcto uso del programa se debe de ingresar la dirección del archivo en la siguiente línea del programa:

```
List<string> txt = new List<string>();  
string filePath = @"C:\Users\Roberto Moya\Desktop\ProyectoAutomatas\proyecto\docs\GRAMATICA.txt";  
  
int a = 0;  
// Abre el archivo utilizando StreamReader  
using (StreamReader reader = new StreamReader(filePath))  
{
```

Imagen No.13 – Dirección del archivo

Una vez cambiada la dirección del archivo deseado al programa se puede ejecutar el programa.

Una vez en ejecución, el programa mostrará el resultado del análisis línea por línea.



```
C:\Users\Roberto Moya\Desktop\ProyectoAutomatas\proyecto\bin\Debug\net6.0\proyecto.exe  
SET valido linea 2  
SET valido linea 3  
SET valido linea 4  
TOKEN valido linea 6  
TOKEN valido linea 7  
TOKEN valido linea 8  
TOKEN valido linea 9  
TOKEN valido linea 10  
TOKEN valido linea 11  
TOKEN valido linea 12  
TOKEN valido linea 13  
TOKEN valido linea 14  
TOKEN valido linea 15  
TOKEN valido linea 16  
TOKEN valido linea 17  
TOKEN valido linea 18  
TOKEN valido linea 19  
TOKEN valido linea 20  
TOKEN valido linea 21  
TOKEN valido linea 22  
TOKEN valido linea 23  
TOKEN valido linea 24  
TOKEN valido linea 25  
TOKEN valido linea 26  
TOKEN valido linea 27  
TOKEN valido linea 28  
TOKEN valido linea 29  
TOKEN valido linea 30  
TOKEN valido linea 31  
TOKEN valido linea 32  
TOKEN valido linea 33  
TOKEN valido linea 34  
TOKEN valido linea 35  
TOKEN valido linea 36  
ACTION valido linea 40  
ACTION valido linea 41  
ACTION valido linea 42  
ACTION valido linea 43  
ACTION valido linea 44  
ACTION valido linea 45  
ACTION valido linea 46  
ACTION valido linea 47  
ACTION valido linea 48  
ACTION valido linea 49  
ACTION valido linea 50  
ACTION valido linea 51  
ACTION valido linea 52  
ACTION valido linea 53  
ACTION valido linea 54  
ACTION valido linea 55
```

Imagen No.14 – Resultado Archivo correcto

El resultado mostrará que fue correcto.

Cuando encuentra un error en el archivo, mostrará un mensaje de error.

```
C:\Users\Roberto Moya\Desktop\ProyectoAutomatas\proyecto\bin\Debug\net6.0\proyecto.exe
Error en línea: 2
Error en línea: 3
Error en línea: 4
TOKEN valido linea 7
TOKEN valido linea 8
TOKEN valido linea 9
TOKEN valido linea 10
TOKEN valido linea 11
TOKEN valido linea 12
TOKEN valido linea 13
TOKEN valido linea 14
TOKEN valido linea 15
TOKEN valido linea 16
TOKEN valido linea 17
Error en la línea 18
TOKEN valido linea 19
TOKEN valido linea 20
TOKEN valido linea 21
ACTION valido linea 25
ACTION valido linea 26
ACTION valido linea 27
ACTION valido linea 28
ACTION valido linea 29
ACTION valido linea 30
ACTION valido linea 31
ACTION valido linea 32
ACTION valido linea 33
ACTION valido linea 34
ACTION valido linea 35
ACTION valido linea 36
```

Imagen No.15 – Errores

Nótese que que al recorrer todo el archivo, muestra los errores que ocurrieron.