

Proyecto 1

Lenguajes Formales y Autómatas

En la primera fase del proyecto es necesario la lectura de un archivo de texto llamado: GRAMATICA.txt el cual contiene la definición de la gramática.

Dicho archivo está compuesto de las siguientes partes:

1. SETS: Contiene la definición abreviada de un conjunto de símbolos terminales, esta parte puede o no venir dentro del archivo, no es necesario que aparezca, pero si aparece, debe poseer al menos un SET.

- a. Ejemplo

SETS

LETRA = 'A'..'Z'+ 'a'..'z'+ '_'

DIGITO = '0'..'9'

CHARSET = CHR(32)..CHR(254)

- b. Tomar en cuenta las siguientes características:

- i. La palabra SETS debe estar en mayúscula.
 - ii. Los sets pueden estar concatenados a través del signo "+", como muestra el set: LETRA.
 - iii. Se puede utilizar la función CHR como lo muestra el set: CHARSET.
 - iv. Puede haber muchos espacios en blanco entre el identificador, el símbolo "=" y la definición.
 - v. Puede haber varios saltos de línea (Enters) entre un SET y otro.
2. TOKENS: Los tokens representan los símbolos terminales y no terminales de la gramática, en esta fase no nos importa si un identificador ha sido declarado o no en los SETS,

- a. Ejemplo

TOKENS

TOKEN 1= DIGITO DIGITO *

TOKEN 51 = ':'

TOKEN 3= LETRA (LETRA | DIGITO)* {

RESERVADAS() }

- b. LA PALABRA TOKENS debe existir y estar en mayúscula
 - c. Esta sección debe existir
 - d. Cada token debe poseer la palabra: TOKEN y un número, seguido del signo igual "=".
 - e. Después del signo igual debe venir una expresión regular, que puede ser uno o varios caracteres (Encerrados en apóstrofes).
 - f. Los signos utilizados para las operaciones de las expresiones regulares son los únicos que no necesitan estar entre comillas, a menos que se quiera denotar su uso como signo terminal.
 - i. Los signos de operaciones para las expresiones regulares son: + * ? () |
3. ACTIONS: La palabra ACTIONS contiene definición de funciones, en este caso específico las palabras reservadas del lenguaje, es importante que la función: Reservadas() siempre debe existir y puede haber otras funciones.

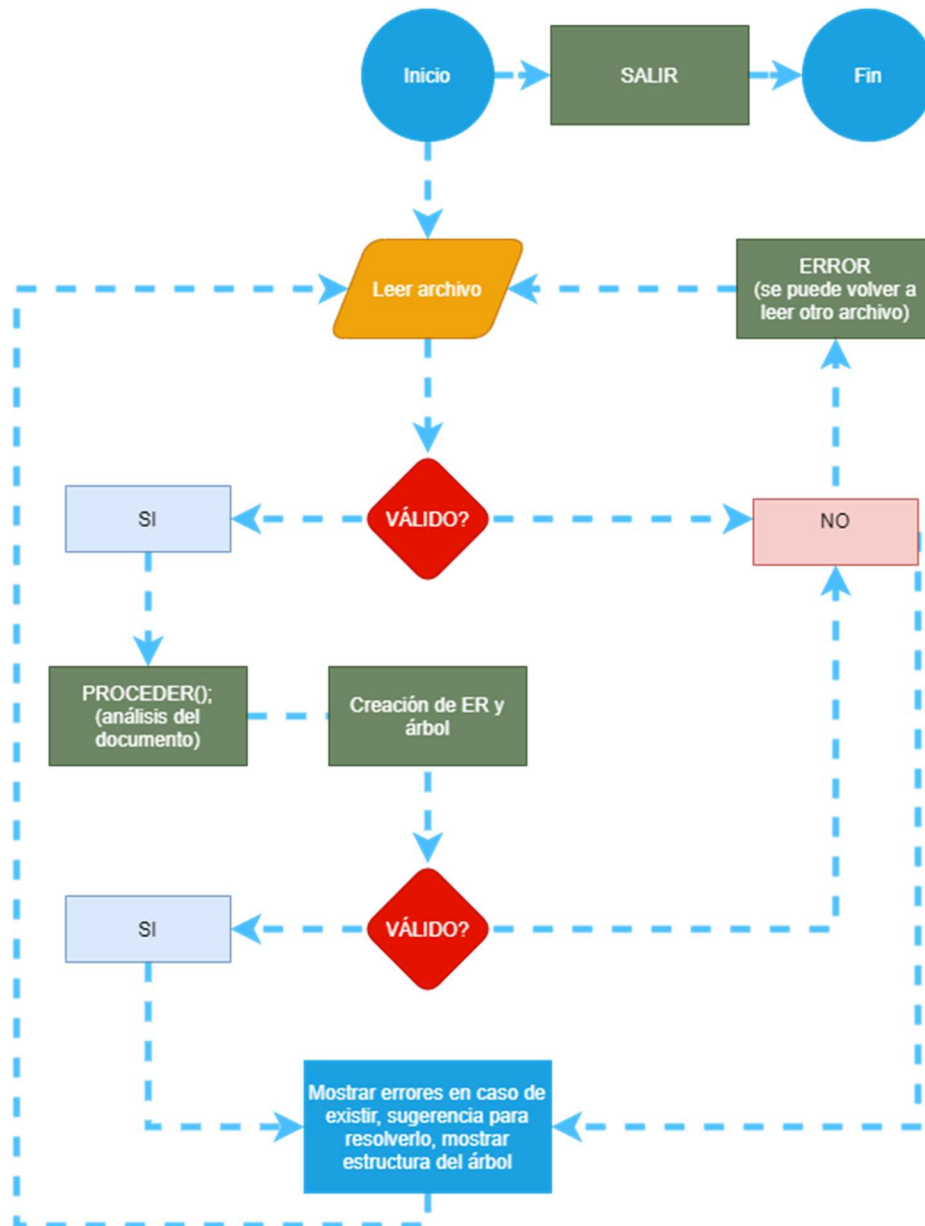
a. Ejemplo

```
ACTIONS
RESERVADAS()
{
    18 = 'PROGRAM'
    19 = 'INCLUDE'
    20 = 'CONST'
    39 = 'DOWNT0'
}
```

- b. La palabra ACTIONS siempre debe venir acompañada de la función RESERVADAS ().
 - c. Todas las funciones deben tener un identificador y unos paréntesis abierto y cerrado.
 - d. Las funciones descritas en ACTIONS deben iniciar y finalizar con llaves {}.
 - e. Los tokens dentro están conformados por: número, signo igual y luego el identificador entre apóstrofes
4. ERROR: La definición de errores debe venir al menos uno, el ERROR debe tener asignado un número, y el identificador debe tener como sufijo la palabra ERROR en mayúscula:
- a. Ejemplo:

ERROR = 54
 - b. Los identificadores solo deben poseer letras, y en la parte derecha del símbolo igual, solamente puede haber números.

Diagrama de flujo



Expresiones regulares:

Terminales =

```
"((?i)SETS|TOKENS|ACTIONS(?-i))([\W \W]+|)$")
```

Variables =

```
"((([A-Za-z])+ ( *)=(( |)*((( *|(\+))(')(.)(')(.)(')(( *)*|(( ( *|(\+?
*)(')(.)(')(.)(')(.)(')(( *)*(\+)(( *)*(')(.)(')(.)(')(.)(')(( *)*|((
+))(')(.)(')(( *)*|(( |)* (CHR)(\(\)[0-9]+(\+)).. (CHR)(\(\)[0-9]+(\+)( |)**(|)$"))
```

Tokens =

```
"(TOKEN)( | )*(\d)+( | )*= ( | )*((((\(+ ( | )*[A-Z]+( | )*[A-Z]+( | )*(\+)+( | )*)+|((( | ))*((\' + (.) + (\' )+)+( | )*)+)|((([A-Z]*)+( | ))*((\*|\||\(\|\)\|\{\|\}\)))+( | ))*|([A-Z]+( | )*[A-Z]+( | ))*((\*|\||\(\|\)\|\{\|\}\)))|\}\)+")
```

Funciones =

```
@"([A-Z])+(\(\\) )*)|( )*(\{|\}))( )*";
```

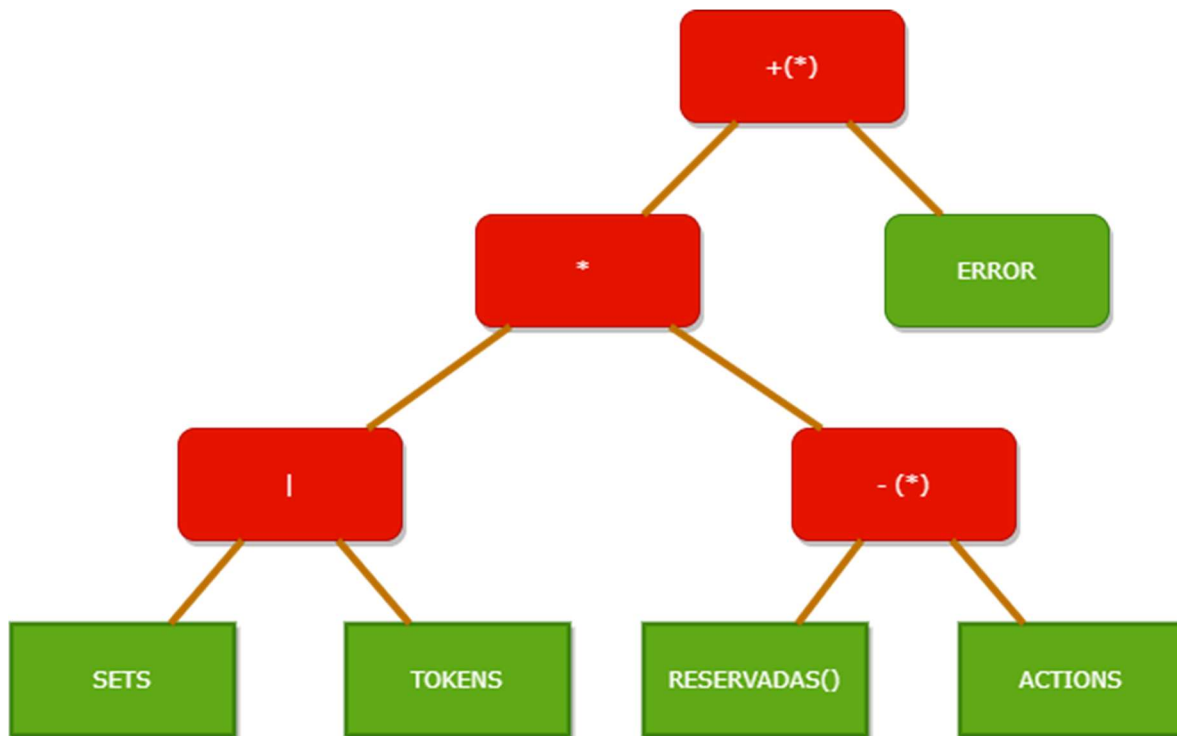
Errores =

```
@"(ERROR)( )*=( )*([0-9]+)( )*";
```

Tokens Funciones

```
@'([0-9])+ ( )*= ( )* ((')([A-Z]+)(')) ( *|))| ( )*((\{)|(\}))( )*');
```

Árbol (GRAMATICA.EXE)



"ERROR", "+", "ACTIONS", "-", "RESERVADAS()", "*", "TOKENS", "|", "SETS"

Parte 2

Lógica de operación

Tomar tokens:

```
nuevoToke = generar.Generar_ER_Tokens(lineas, inicioTokens, finalTokens, erMatch);
```

Tomar sets

```
erSets = generar.Generar_ER_Sets(lineas, inicioSets, finalSets);
```

Hacer march con los tokens y sets

```
erMatch = generar.Hacer_Match(lineas, inicioTokens, finalTokens, erMatch);
```

Verificar que los tokens esten declarados

```
if (erMatch[j].Contains(erSets[i]))
```

Tomar los tokens e iniciar con (y finalizar con)#

```
generar.Generar_ER_Tokens(lineas, inicioTokens, finalTokens, erMatch);
```

Separar los simbolos de los tokens

```
Proceso.Dividir(nuevoToke);
```

Tomarlos del árbol para un recorrido infijo

```
proceso.Recorrer(entrada);
```

Establecer orden de los signos

```
if (cadenaT[i].ToString() != "." && cadenaT[i].ToString() != "(" &&  
cadenaT[i].ToString() != "|" && cadenaT[i].ToString() != ")" &&  
cadenaT[i].ToString() != "*" && cadenaT[i].ToString() != "+" &&  
cadenaT[i].ToString() != "?") en Obtener(ArrayList cadenaT)
```

Manejo del árbol, inicializar las listas, colas y grupos

```
Queue<string> grupo1 = new Queue<string>();  
List<string> grupo2 = new List<string>();  
string[] conjuntos = new string[ST.Count + 1];  
string temp = "";  
Nodo N2 = Simbolos.Pop();
```

Inicio de shunting yard (solo con los datos ya que los símbolos y la jerarquía se toman en el orden de los signos que se obtuvieron)

```
while (grupo1.Count != 0)  
if (grupo1.Count != 0)
```

```
NuevoNodo(string op, ref Stack<Nodo> simb) → ObtenerFollow(Nodo nuevo) →  
ObtenerFollowSig(Nodo nuevo, Nodo nuevo2)
```

Representación

Tabla datagrid

```
TablaDgrd(ArrayList cadena1, List<string> grupo2)
```

(Toma la lista de datos y se les asigna un a los grupos identificadores de tipo char)

Se lee los datos que se obtuvieron de la producción de la tabla y se comparan con los puntos establecidos en el algoritmo shunting yard también se imprimen los Follows

Follows

```
Imprimir()
```

En el proceso de obtener los nodos se van mostrando en conjunto los First y Last

First y Last

```
NuevoNodo(string op, ref Stack<Nodo> simb)
```

Precedencia de los datos:

```
case '*': Orden = 4; break;
case '+': Orden = 4; break;
case '?': Orden = 4; break;
case '.': Orden = 3; break;
case '|': Orden = 2; break;
case '(': Orden = 1; break;
```

Parte 3

Lógica de operación

PROGRAMA PRINCIPAL

Exportamos la tabla de transiciones, los tokens y las palabras reservadas a la carpeta del segundo programa

```
if (existe)
{
    File.Copy(@"Datagrid.txt", copiarTXT,true);
    File.Copy(@"SonTokens.txt", copiarTXT2, true);
    File.Copy(@"SonReservadas.txt", copiarTXT3, true);
    Copiar(copiar,guardar);
}
```

Copiamos la carpeta donde se aloja el segundo programa a donde el usuario lo indique

```
if (existe)
{
    File.Copy(@"Datagrid.txt", copiarTXT,true);
    File.Copy(@"SonTokens.txt", copiarTXT2, true);
    File.Copy(@"SonReservadas.txt", copiarTXT3, true);
    Copiar(copiar,guardar);
}
```

SEGUNDO PROGRAMA

Cargamos los archivos exportados del programa principal para su análisis

```
LeerTokens();
```

Haciendo uso de un textbox cargamos el texto que queremos analizar

```
SepararEntradaEvaluar(entradaText);
```

Dado ciertas reglas, se separa el texto con espacio para después analizar si pertenece a la tabla de transiciones

```
SepararEntradaEvaluar(string entrada)
```

Verifica los caracteres comparándolo con la tabla de transiciones

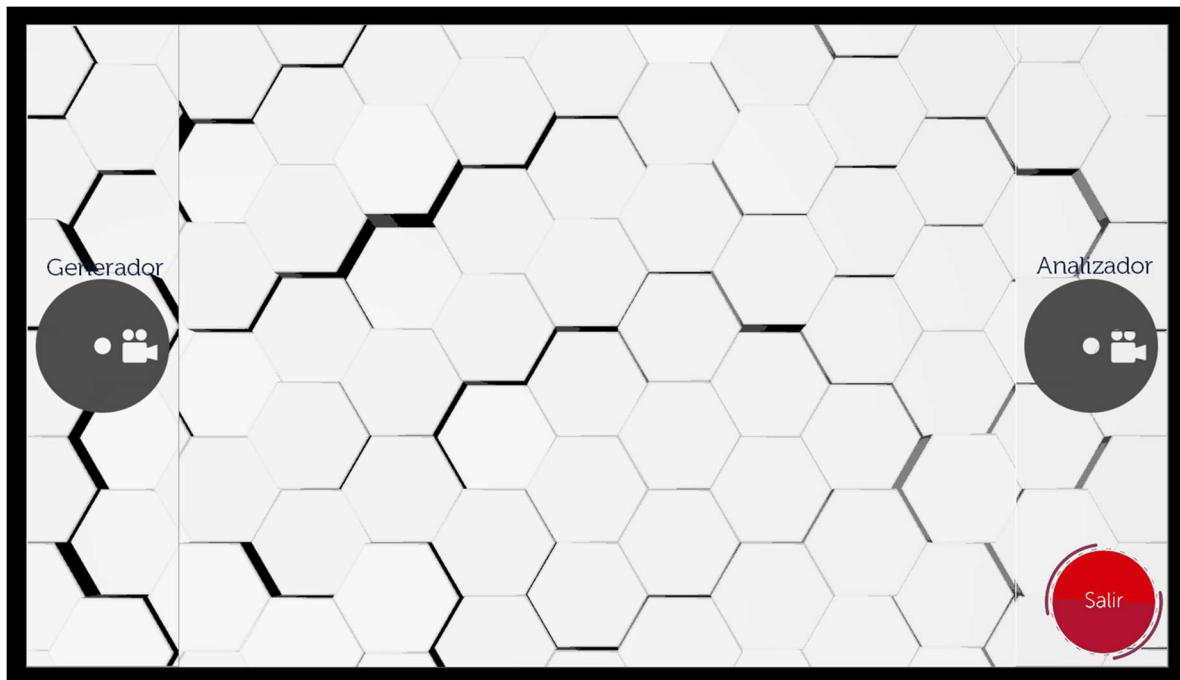
```
VerificarEntrada(string[] entrada)
```

Si la cadena dada es correcta, ahora se analiza a que token pertenecen los caracteres dados en el texto dado

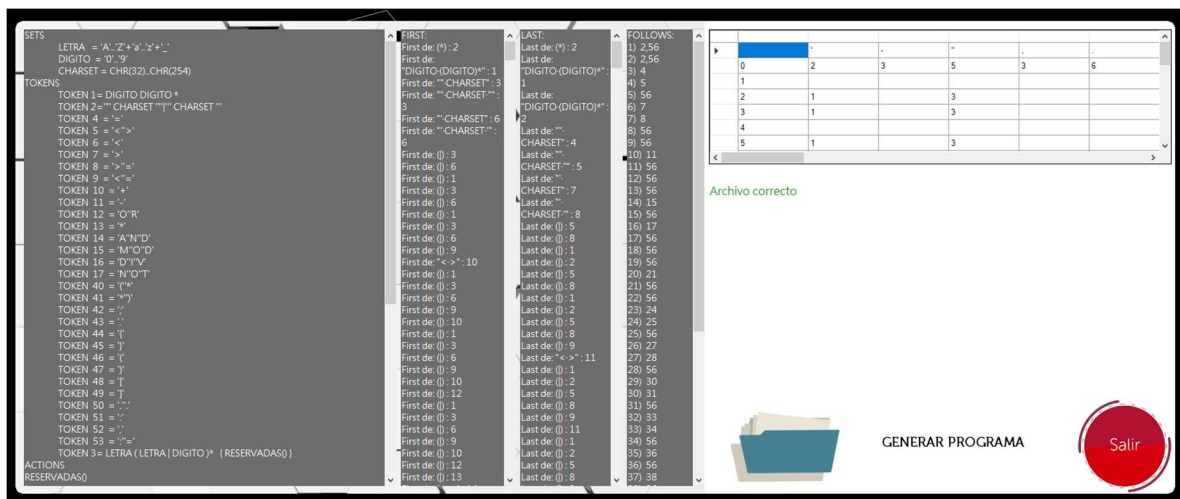
```
SepararEntradaDemostrar(entradaText);
```

Representación visual

Manual de usuario



Generador (Correcto)



Generador (Incorrecto)

The interface displays a list of tokens on the left, categorized under 'SETS' and 'TOKENS'. The 'SETS' section includes 'DIGITO = 0..9' and 'CHARSET = CHR(32)..CHR(254)'. The 'TOKENS' section lists 53 tokens, including 'TOKEN 1 = DIGITO DIGITO +', 'TOKEN 2 = "" CHARSET "" CHARSET ""', 'TOKEN 4 = "="', 'TOKEN 5 = "<" ">"', 'TOKEN 6 = "<"', 'TOKEN 7 = ">"', 'TOKEN 8 = ">" "="', 'TOKEN 9 = "<" "="', 'TOKEN 10 = "+"', 'TOKEN 11 = "."', 'TOKEN 12 = "O" "R"', 'TOKEN 13 = "P"', 'TOKEN 14 = "A" "N" "D"', 'TOKEN 15 = "M" "O" "D"', 'TOKEN 16 = "D" "I" "V"', 'TOKEN 17 = "N" "O" "T"', 'TOKEN 40 = "I" "M"', 'TOKEN 41 = "J"', 'TOKEN 42 = "V"', 'TOKEN 43 = "I"', 'TOKEN 44 = "T"', 'TOKEN 45 = "T"', 'TOKEN 46 = "T"', 'TOKEN 47 = "J"', 'TOKEN 48 = "T"', 'TOKEN 49 = "T"', 'TOKEN 50 = "I" "I"', 'TOKEN 51 = "I"', 'TOKEN 52 = "I"', 'TOKEN 53 = "I" "="', and 'TOKEN 3 = LETRA (LETRA | DIGITO)* (RESERVADAS)'. The 'ACTIONS' section is labeled 'RESERVADAS()'. On the right, there are columns for 'FIRST', 'LAST', and 'FOLLOWS'. A message 'TOKEN NO DECLARADO EN SETS' is displayed in green, and 'DEBE DECLARAR UN SET PARA EL TOKEN' is displayed in red. A red circular button labeled 'Salir' is in the bottom right corner.

Analizador (Correcto)

The interface shows a text input field at the top containing the word 'Program'. Below the input field, the word 'CORRECTO' is displayed in a green box. Further down, the text 'Program --> 5' is shown. In the bottom right corner, there is a dark circular button labeled 'Salir'.

Analizador (Incorrecto)

416854pPROGRAM

INCORRECTO

Salir