Universidad Rafael Landívar

Facultad de Ingeniería

Ingeniería en Informática y Sistemas

Lenguajes Formales y Autómatas

Ingeniero Moisés Alonso

***Proyecto fase 1***

***Generador de Scanner***

Albarizaez Jose Eduardo

Carne No. 1106918

Fecha de Entrega: Guatemala, 4 de marzo de 2020

**Índice**

**Contenido Pg.**

[**Pseudocódigo** 2](#_Toc34145083)

[**Expresiones Regulares** 10](#_Toc34145084)

[**Árboles de expresión** 11](#_Toc34145085)

[**Discusion de Resultados** 12](#_Toc34145086)

[**Conclusiones** 12](#_Toc34145087)

# **Pseudocódigo**

switch (Validacion)

{

case 0:

linea = linea.Trim(' ');

if (linea == "SETS")

{

Validacion++;

}

else if (linea == "TOKENS")

{

Validacion = 2;

}

else

{

Error = true;

}

break;

case 1:

if (linea.Contains("TOKEN") == false)

{

var comp = evaluar.TokenSets(linea);

if (comp == null)

{

Error = true;

}

Contador++;

}

else

{

if (Contador > 0)

{

linea = linea.Trim(' ');

if (linea == "TOKENS")

{

Validacion++;

Contador = 0;

}

else

{

Error = true;

}

}

else

{

Error = true;

}

}

break;

case 2:

if (linea.Contains("TOKEN"))

{

Contador++;

}

else

{

if (Contador > 0)

{

linea = linea.Trim(' ');

if (linea == "ACTIONS")

{

Validacion++;

Contador = 0;

}

else if (linea == "")

{ }

else

{

Error\_tipo = "La linea de comando 'ACTIONS' no se encuentra o error al definir token";

Error = true;

}

}

else

{

Error\_tipo = "No venia especificado ningun TOKEN";

Error = true;

}

}

break;

case 3:

if (contadorActions == 0)

{

Expresiones actions = new Expresiones();

if (actions.analizarActions(linea.Trim(' ')) && totalActions == 0 && linea.Trim(' ') == "RESERVADAS()")

{

contadorActions++;

}

else if (totalActions > 0)

{

if (actions.analizarActions(linea.Trim(' ')))

{

contadorActions++;

}

else if (linea.Contains("ERROR"))

{

Validacion++;

var comp = evaluar.TokenError(linea);

if (comp == null || comp.Count != 3)

{

Error\_tipo = "Error en la sintaxis de linea";

Error = true;

}

contadorErrores++;

Contador++;

}

else if (actions.analizarActions(linea) == false)

{

Error = true;

}

}

else

{

Error = true;

}

}

else if (contadorActions == 1)

{

if (linea.Trim(' ') == "{")

{

contadorActions++;

}

else

{

Error = true;

}

}

else if (contadorActions == 2)

{

if (linea.Trim(' ') == "}")

{

if (Contador == 0)

{

Error = true;

}

else

{

totalActions++;

contadorActions = 0;

}

}

else if (linea == "" || linea == " " || linea == "\t")

{

}

else if (linea.Contains("ERROR") == false)

{

var comp = evaluar.TokenActions(linea);

if (comp == null || comp.Count != 5)

{

Error = true;

}

Contador++;

}

else

{

Error = true;

}

}

else

{

if (linea.Contains("ERROR"))

{

contadorErrores++;

}

}

break;

case 4:

if (contadorErrores > 0)

{

var comp = evaluar.TokenError(linea);

if (comp == null || comp.Count != 3)

{

Error = true;

}

contadorErrores++;

Contador++;

}

else

{

Error = true;

}

break;

}

}

if (Error == true)

{

break;

}

}

if (totalActions > 0 && contadorErrores == 0 && Error != true)

{

Error = true;

linea\_Archivo++;

}

if (Error == true)

{

WriteLine("Error en la linea " + Convert.ToString(linea\_Archivo) + " Posible error: " + Error\_tipo);

}

else

{

WriteLine("Lectura realizada exitosamente");

}

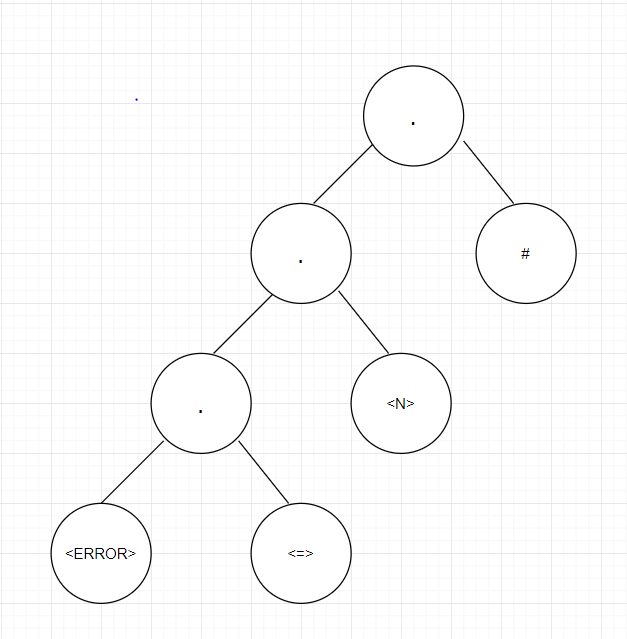
WriteLine("------------------------------------------------------------------------------------------------------------");

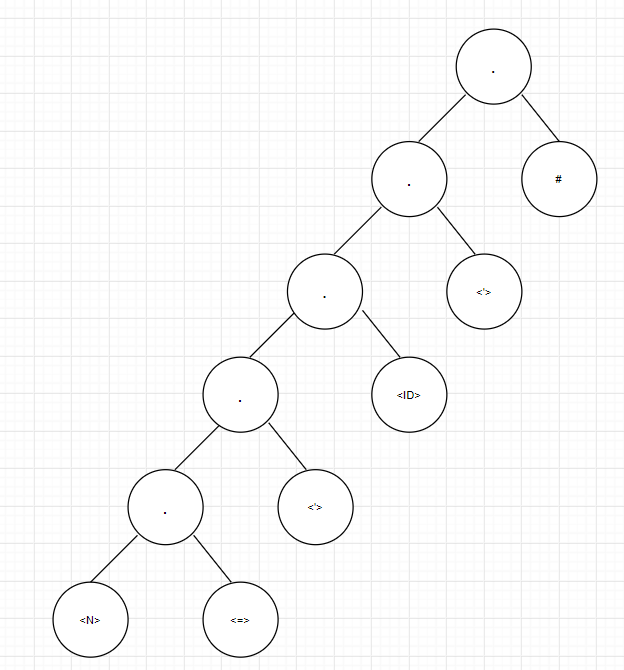
ReadKey();

# **Expresiones Regulares**

|  |  |
| --- | --- |
| Sets | (<ID>.<=>.((<'>.<C>.<'>)|(<CHR>.<(>.<N>.<)>)).((<..>|<+>).((<'>.<C>.<'>)|(<CHR>.<(>.<N>.<)>)))\*) |
| Tokens | (<TOKEN>.<N>.<=>(<ID>|<'>.<C>.<'>|<\*>|<|>|<+>|<(>|<)>|<{>|<}>+) |
| Actions | (<N>.<=>.<'>.<ID>.<'>) |
| Errores | (<ERROR>.<=>.<N>) |

# **Árboles de expresión**





# **Planteamiento**

En mi caso realice unos cuantos cambios respecto a como plantear mis expresiones regulares, para eso tome en consideración un ejemplo realizado en clase en el cual mostraban un símbolo terminal de la siguiente manera “<ID>” con lo cual tuve la idea de plantear que todos los símbolos terminales que yo iba a utilizar en mi expresión regular vendrían planteados de la manera en que todo lo que este dentro de los símbolos <> se convertirá en símbolo terminal, con esto mis expresiones regulares quedaron de una manera mas sencilla y me permitió realizar métodos que pudieran alimentar el árbol de expresiones con un string.

# **Conclusiones**

1. Las expresiones regulares nos permiten validar texto.
2. Las expresiones regulares son utilizadas de gran manera con librerías como regex.
3. Los árboles poseen varias aplicaciones y entre ellas esta el de evaluar expresiones regulares.