Universidad Rafael Landívar

Facultad de Ingeniería

Lenguajes Formales y Autómatas, sec 01

Mgtr. Moises Alonso

Proyecto Práctico Fase: 1

José Alejandro Montenegro Monzón

1229918

Guatemala 02 de febrero de 2020

**ÍNDICE**

1. **Entradas, Procesos, Salidas y PseudoCódigo…………………………… 3**
   1. **Entrada………………………………………………………………… 3**
   2. **Proceso………………………………………………………………… 3**
   3. **Salida………………………………………………………………….. 3**
   4. **Pseudocódigo………………………………………………………….. 3**
   5. **Pseudocódigo Método RecorridoArbolACTIONS()………………… 5**
   6. **Diagrama De Flujo……………………………………………………. 7**
      1. **Diagrama De Flujo …………………...……………………… 7**
2. **Terminologías a Utilizar………………………………………………….. 7**
3. **Expresiones y Arboles de Expresiones…………………………………… 8**
4. **MANUAL DE USUARIO………………………………………………… 10**
5. **Entradas, Procesos, Salidas y PseudoCódigo**
6. **Entrada:** La única entrada de este proyecto es la lectura del archivo de texto. Si el archivo de texto no recibe un archivo de texto, sino recibe otra clase de archivo, éste no continuará la ejecución.
7. **Salida:** La salida será la validación de la estructura del archivo de texto.
8. **Proceso:**

* Separar por listas cada sección del Archivo (SETS, TOKENS, ACTIONS, ERROR)
* Crear los árboles con sus respectivas expresiones
* Recorrer cada frase que contendrá cada una de la lista y recorrer el árbol en In-Orden
* Recorrer cada Nodo y validar cada una de las frases en las listas.
* Si el resultado de cada

1. **Pseudocódigo método Main()**

Inicio

Main()

Read(String D\_Archivo);

If(Read[D\_Archivo.Count()-1]==”txt)

{

List listaSETS = New List();

List listaTOKENS = New List();

List listaACTIONS = New List();

List listaERROR = New List();

Read(archive, ref listaSETS, ref listaTOKENS, ref listaACTIONS, ref listaERRORS);

If(ListaSETS.Count()!=0)

{

Foreach(string linea in listaSETS)

{

RecorrerArbol(linea, A,B,C,…);

If(error)

{

Write(“La línea x contiene un error);

Break;

}

}

}

Else

{

If(error)

{

Console.ReadLine();

}

Else

{

IF(ListaTOKENS.Count()!=0)

{

Foreach(string linea in listaTOKENS)

{

RecorrerArbol(linea, A,B,C,…);

If(error)

{

Write(“La línea x contiene un error);

Break;

}

}

If(error)

{

ReadLine();

}

Else

{

If(ListaACTIONS()!=0&&ListaACTIONS.Countains(“RESERVADAS())

{

Foreach(string linea in listaTOKENS)

{

RecorrerArbol(linea, A,B,C,…);

If(error)

{

Write(“La línea x contiene un error);

Break;

}

}

If(error)

{

Read();

}

Else

{

If(ListaSETS.Count()!=0)

{

Foreach(string linea in listaSETS)

{

RecorrerArbol(linea, A,B,C,…);

If(error)

{

Write(“La línea x contiene un error);

Break;

}

}

If(error)

{

Read();

}

Else

{

Write(“Su archivo no contiene ningún error”);

}

}

}

}

Else

{

Write(“Su método no contiene los parámetros requeridos!);

Read();

}

}

}

Else

{

Write(“Su archive no contiene el método TOKENS correcto”);

}

}

}

}

Else

{

Write(“El archivo que ingresó no es un archivo de texto);

}

Fin

1. **Pseudocódigo Método RecorridoArbolACTIONS()**

public void RecorrerActions(Node NodoActual, Node PadreAux, bool encontrar, ref string frase, ref string Continuar, string N, char B, string A, ref int cantidad)

{

if (NodoActual == null)

{

return;

}

RecorrerActions(NodoActual.hijoIZ, PadreAux, encontrar, ref frase, ref Continuar, N, B, A, ref cantidad);

var R = false;

EncontrarPadre(NodoActual, Raiz, ref PadreAux, ref R);

var contador = 0;

if (frase != string.Empty)

{

switch (PadreAux.elemento.caracter)

{

case ".":

if (NodoActual.elemento.caracter == "=")

{

encontrar = EncontrarCaracterAuxActions(NodoActual.elemento.caracter, frase[0], N, B, A);

if (encontrar)

{

frase = frase.Substring(1);

contador++;

}

Continuar = (contador == 1) ? "PC" : "NPC";

}

else

{

if (NodoActual.elemento.caracter == "'")

{

encontrar = EncontrarCaracterAuxActions(NodoActual.elemento.caracter, frase[0], N, B, A);

if (encontrar)

{

frase = frase.Substring(1);

contador++;

}

Continuar = (contador == 1) ? "PC" : "NPC";

cantidad = Continuar == "PC" ? cantidad + 1 : cantidad;

}

}

break;

case "+":

do

{

encontrar = EncontrarCaracterAuxActions(NodoActual.elemento.caracter, frase[0], N, B, A);

if (encontrar)

{

frase = frase.Substring(1);

contador++;

}

if (frase.Length == 0)

{

break;

}

}

while (encontrar == true);

Continuar = (contador >= 1) ? "PC" : "NPC";

break;

case "\*":

do

{

encontrar = EncontrarCaracterAuxActions(NodoActual.elemento.caracter, frase[0], N, B, A);

if (encontrar)

{

frase = frase.Substring(1);

contador++;

}

}

while (encontrar == true);

Continuar = (contador >= 0) ? "PC" : "NPC";

break;

}

}

if (Continuar == "PC")

{

RecorrerActions(NodoActual.hijoDR, PadreAux, encontrar, ref frase, ref Continuar, N, B, A, ref cantidad);

}

else

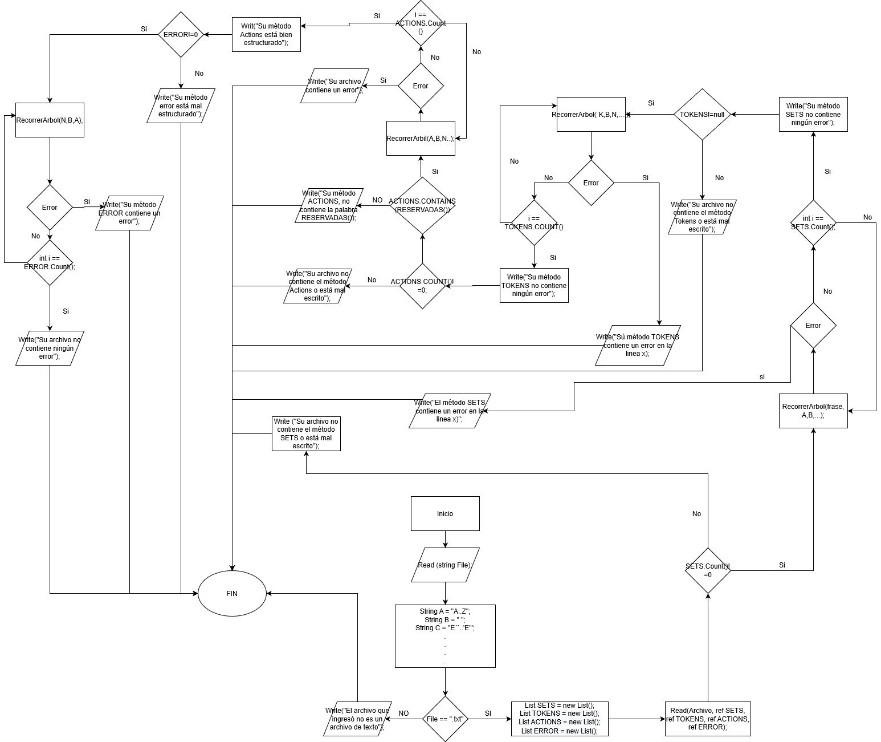
{

return;

}

}

1. **Diagrama de flujo**
2. **Diagrama De Flujo del método Main()**



1. **Terminologías a Utilizar**

A = "ABCDEFGHIJKLMNÑOPQRSTUVWXYZ";

B = ' ';

C = "'E'..'E'";

D = "'E'";

E = string.Empty;

for (int po = 0; po < 256; po++)

{

E += Convert.ToChar(Convert.ToByte(po));

}

F = "CHR(N)";

G = "CHR(NN)";

H = "CHR(NNN)";

J = "CHR(N)..CHR(N)";

L = "CHR(N)..CHR(NN)";

M = "CHR(N)..CHR(NNN)";

N = "0123456789";

Ñ = "CHR(NN)..CHR(N)";

O = "CHR(NN)..CHR(NN)";

Q = "CHR(NN)..CHR(NNN)";

R = "CHR(NNN)..CHR(N)";

S = "CHR(NNN)..CHR(NN)";

T = "CHR(NNN)..CHR(NNN)";

P = "+";

X = '"';

I = E + X;

U = "'I'";

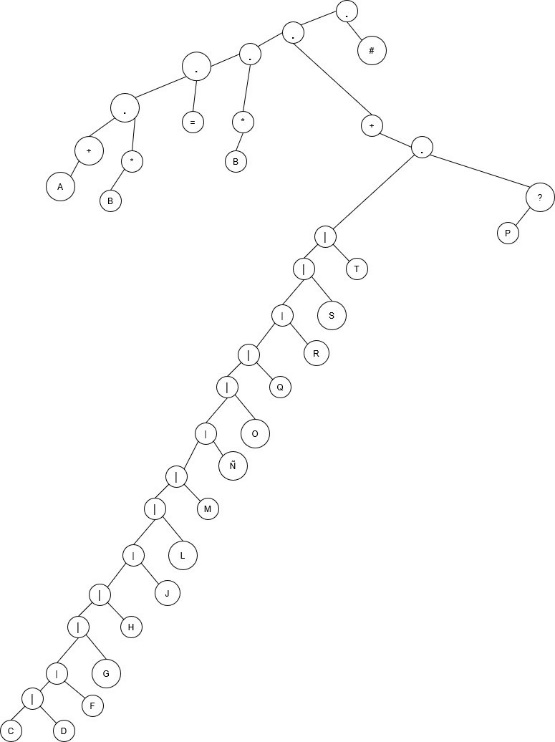
K = "TOKEN";

V = " ";

W = "+\*/()[]{}|?";

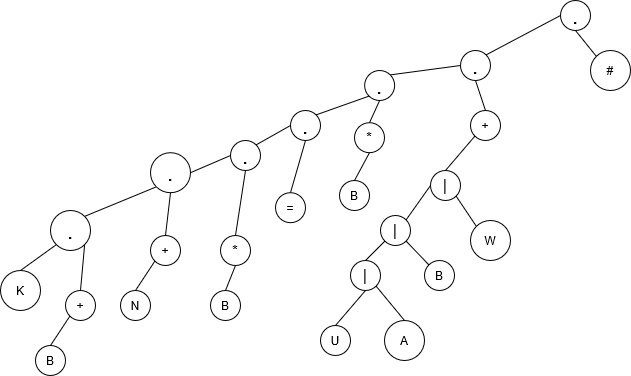
1. **Expresiones y Arboles de Expresiones**
2. **Expresión SETS y Árbol de Expresión SETS**

ExpSets = "((A+.B\*.=.B\*).((C|D|J|L|M|Ñ|O|Q|R|S|T|H|F|G).B\*.P?.B\*)+).#"



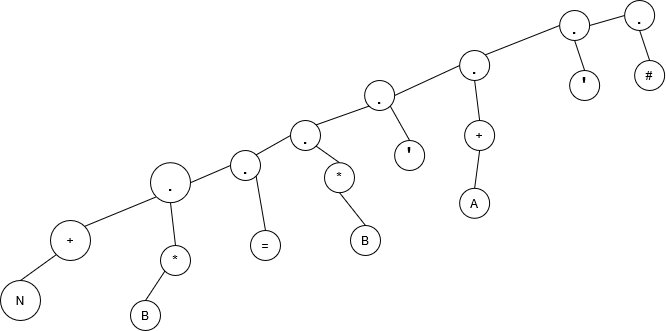
1. **Expresión TOKENS y Árbol de Expresión TOKENS**

ExpTokens = "((K.B+.N+.B\*.=.B\*).(U|A|B|W)+).#";



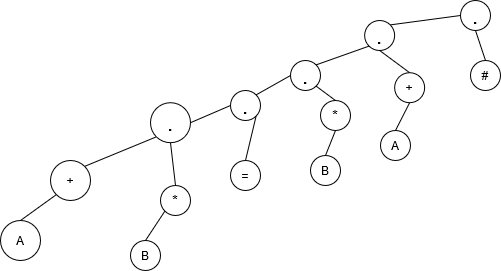
1. **Expresión ACTIONS y Árbol de Expresión ACTIONS**

ExpActions = "(N+.B\*.=.B\*.'.A+.').#";



1. **Expresión ERROR y Árbol de Expresión ERROR**

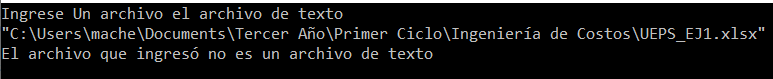
ExpErrors = "(A+.B\*.=.B\*.N+).#";



1. **MANUAL DE USUARIO**
   1. **Lectura Archivo**

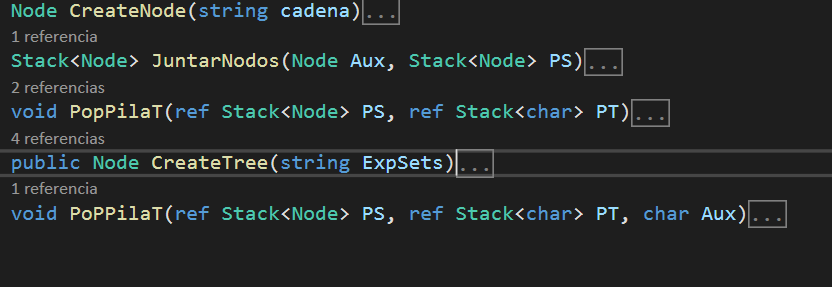


El usuario debe de ingresar el archivo de texto que desea leer. Si el usuario no ingresa un archivo de texto con la terminación .txt, sino que ingresa la dirección de otro tipo de archivo el programa indicará que el archivo es invalido y la ejecución del programa finalizará.



Tras haber validado que el archivo ingresado sea un archivo de texto, se procede a la lectura del archivo. Para esta parte se van a separar los métodos en 4 listas diferentes. Una para SETS, una para TOKENS, una para ACTIONS y una para ERROR.

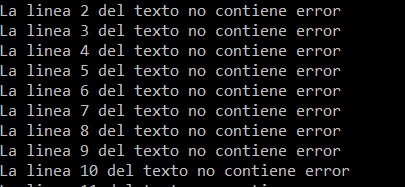
* 1. **Creación Árbol**



Tras haber terminado la lectura del archivo de texto (.txt) que se ingresó, se procede a crear el Árbol de expresiones. Se utiliza estos cuatro métodos para poder crearlo.

* 1. **Recorrido Árbol**

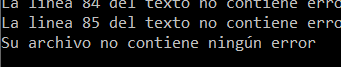
El recorrido del árbol se realiza de forma en IN-ORDEN. Se ingresa la frase entera al recorrido y se valida carácter por carácter que el recorrido no contenga ningún error. En el caso de que el recorrido no sea incorrecto, el programa indicará que la línea “n” no contiene error.



En el caso contrario, el programa indicará en que línea se encuentra el error. NOTA: los números de las líneas solamente indicarán que línea que contenga caracteres es la que está incorrecta. No tomará en cuenta los saltos de línea.



* 1. **Resultado**



El resultado esperado en el caso de que el archivo no contenga ningún error será que en la consola muestre “Su archivo no contiene ningún error”. En el caso de que, durante el recorrido del árbol, se encuentre un error, el programa indicará que está mal la línea “n”.

* 1. **Ejemplo**

El archivo de prueba No. 8 indicará que contiene un error en la línea 85 (línea que contendrá caracteres).



En el caso de la prueba No. 2 el resultado será que el programa indique que el archivo contiene un error en la línea 2;

