Manual técnico

Resumen

En el siguiente manual se emplearán muchos conceptos sobre los compiladores para analizar léxica y sintácticamente un archivo de entrada. Las herramientas para utilizar para realizar dichos análisis serán Jflex y Cup, los cuales nos proporcionan una cantidad de funciones para facilitarnos dicho análisis.

Explicación de funcionalidades del programa:

A continuación se detallan a fondo las funcionalidades del programa explicando las partes más importantes del mismo.

Análisis de Jflex:

Con la sintaxis de Jflex procedemos a importar todas las librerías necesarias para nuestro análisis del archivo.

```
1    package analizadores;
2    import java_cup.runtime.Symbol;
3    import Error.*;
4    import Proyectol_Compi.Menu.*;
5
6    %%
7    %class Lector
8    %public
9    %line
10   %char
11   %cup
12   %unicode
13   %ignorecase
```

Luego establecemos el método principal llamado init.

```
%init{
   yyline = 1;
   yychar = 1;
%init}
```

Seguidamente establecemos todas las expresiones regulares de los tokens que utilizaremos para el análisis sintáctico como se muestra a continuación:

```
20    TKESPECIALES = (\\n|\\\'|\\\")
21    WHITE = [ \r\t]+
22    COMMENT = "<!"[^("!>")]+"!>"|\/\/.+
23    IDENTIFICADOR = [a-zA-Z_][a-zA-Z0-9_]*
24    NUMERO = [0-9]+
25    AZMINUS = [a-z]
26    AZMAYUS = [A-Z]
27    NOTACIONCOMAS = ([a-zA-Z]\,|[0-9]+\,|{TKESPECIALES}+\,)+([a-zA-Z]|[0-9]+|{TKESPECIALES})
28    //POLACA = ((\.|\||\*|\+|\?|[ ])+(\{[a-zA-Z][a-zA-Z0-9_]*\})+|(\.|\||\*|\+|\?|[ ])+(\"[^\"]*\")+|(\{[a-zA-Z][a-zA-Z0-9_]*\})+)+
29    SIMBOLO = [\!\#\$\%\&\(\)\,\-\/\:\;\<\=\>\@\\\[\]\^\_\`\{\}]
30    CADENA = \"{TKESPECIALES}*[^\"]+\"
31
32    //SIMBOLOSESPECIALES = \"(\.|\+|\*|\?|\||\;|\:|%%|\"|\-|\,|\~|\ )\"
33
34    %%
35
36    "{"     {return new Symbol(sym.LLAVELEFT, yytext());}
37    "}"     {return new Symbol(sym.LLAVERIGHT, yytext());}
38    ";"     {return new Symbol(sym.PUNTOYCOMA, yytext());}
39    "%%\n%%" {return new Symbol(sym.CAMBIOSECCION, yytext());}
40    "-"     {return new Symbol(sym.GUION, yytext());}
41    ">"     {return new Symbol(sym.MAYOR, yytext());}
42    ":"     {return new Symbol(sym.DOSPUNTOS, yytext());}
43    "~"     {return new Symbol(sym.ONDULADO, yytext());}
44    "CONJ"  {return new Symbol(sym.CONJ, yytext());}
45
46    "*" {return new Symbol(sym.CERRKLEE, yytext());}
47    "+" {return new Symbol(sym.CERRPOSI, yytext());}
48    "." {return new Symbol(sym.PUNTO, yytext());}
49    "?" {return new Symbol(sym.PREGUNTA, yytext());}
50    "|" {return new Symbol(sym.OR, yytext());}
51
52    \n {yychar = 1;}
53
54    {WHITE} {}
55
56    {COMMENT} {return new Symbol(sym.COMMENT, yytext());}
57    {AZMINUS} {return new Symbol(sym.AZMINUS, yytext());}
58    {AZMAYUS} {return new Symbol(sym.AZMAYUS, yytext());}
59    {TKESPECIALES} {return new Symbol(sym.TKESPECIALES, yytext());}
60    {IDENTIFICADOR} {return new Symbol(sym.IDENTIFICADOR, yytext());}
61    {NUMERO} {return new Symbol(sym.NUMERO, yytext());}
62    {NOTACIONCOMAS} {return new Symbol(sym.NOTACIONCOMAS, yytext());}
63
64    {SIMBOLO} {return new Symbol(sym.SIMBOLO, yytext());}
65    {CADENA} {return new Symbol(sym.CADENA, yytext());}
```

Por último se establece el método que capturará todos los errores léxicos.

```
67    . {
68        System.out.println("\u001B[31m"+"Error lexico en: "+yytext()+" en la linea: "+yyline+ " en la columna: "+ yychar+ "\u001B[0m");
69        Proyectol_Compi.Menu.listaErr.addError(new Error_("Error léxico: "+yytext(), "Lexico", yyline, yychar));
70    }
71
```

Análisis de Cup:

Así como se realizó el análisis de Jflex se realizará también el análisis de Cup, el cual consta de lo siguiente:

Primeramente se importan las librerías necesarias para dicho análisis.

```
1    package analizadores;
2    import java_cup.runtime.*;
3    import java.util.ArrayList;
4    import Proyectol_Compi.Menu.*;
5    import Error.*;
6    import Estructuras.ArbolBinario;
7    import Estructuras.NodoArbol;
8
```

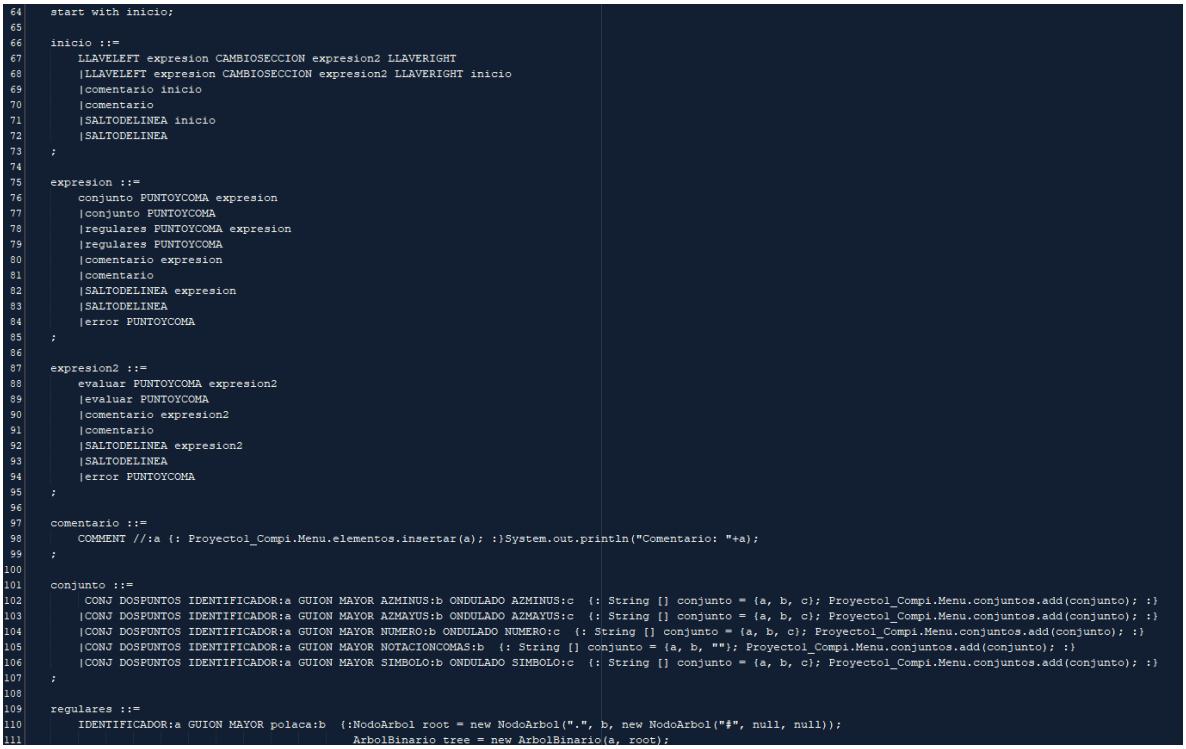
Luego en la parte llamada parser code se capturarán todos los errores sintácticos leídos en el archivo ingresado.

```
9   parser code
10  {:
11          /*Method that is called when parser can be recovered*/
12          public void syntax_error(Symbol s){
13              //System.out.println("\u001B[31m"+"Error sintactico en la linea "+s.left+" columna "+s.right+" No se esperaba este componente: "+s.value+"\u001B[0m");
14              //Instruction.lista.addError(new Error_("Sintactico error: "+s.value, "Sintactico"));
15              Proyectol_Compi.Menu.listaErr.addError(new Error_("Error sintactico: "+s.value, "Sintactico", s.left, s.right));
16      }
17          /*Method that is called when parser can't be recovered*/
18          public void unrecovered_syntax_error(Symbol s) throws java.lang.Exception{
19              //System.out.println("\u001B[31m"+"Error sintactico uncovered en la linea "+s.left+" columna "+s.right+" No se esperaba este componente: "+s.value+"\u001B[0m");
20              //Instruction.lista.addError(new Error_("Sintactico error: "+s.value, "Sintactico"));
21              Proyectol_Compi.Menu.listaErr.addError(new Error_("Error sintactico: "+s.value, "Sintactico", s.left, s.right));
22      }
23  :}
24
```

El siguiente paso es establecer todos los elementos terminales y no terminales, tal como se establecen en una gramática de tipo 2.

```
25    //Parte 1 terminales
26    terminal String COMMENT;
27    terminal String IDENTIFICADOR;
28    terminal String NUMERO;
29    terminal String AZMINUS;
30    terminal String AZMAYUS;
31    terminal String NOTACIONCOMAS;
32    terminal String CADENA;
33    terminal String SIMBOLO;
34    terminal String TKESPECIALES;
35
36    //Parte 2 terminales
37    terminal String LLAVELEFT;
38    terminal String LLAVERIGHT;
39    terminal String PUNTOYCOMA;
40    terminal String CAMBIOSECCION;
41    terminal String GUION;
42    terminal String MAYOR;
43    terminal String DOSPUNTOS;
44    terminal String ONDULADO;
45    terminal String CONJ;
46    terminal String SALTODELINEA;
47
48    terminal String CERRKLEE;
49    terminal String CERRPOSI;
50    terminal String PUNTO;
51    terminal String PREGUNTA;
52    terminal String OR;
53
54    //No terminales
55    non terminal inicio;
56    non terminal expresion;
57    non terminal expresion2;
58    non terminal conjunto;
59    non terminal regulares;
60    non terminal evaluar;
61    non terminal comentario;
62    non terminal NodoArbol polaca;
```

Para por último pasar a la parte de las producciones, como se indicó anteriormente aquí es donde se construye la gramática de tipo 2.

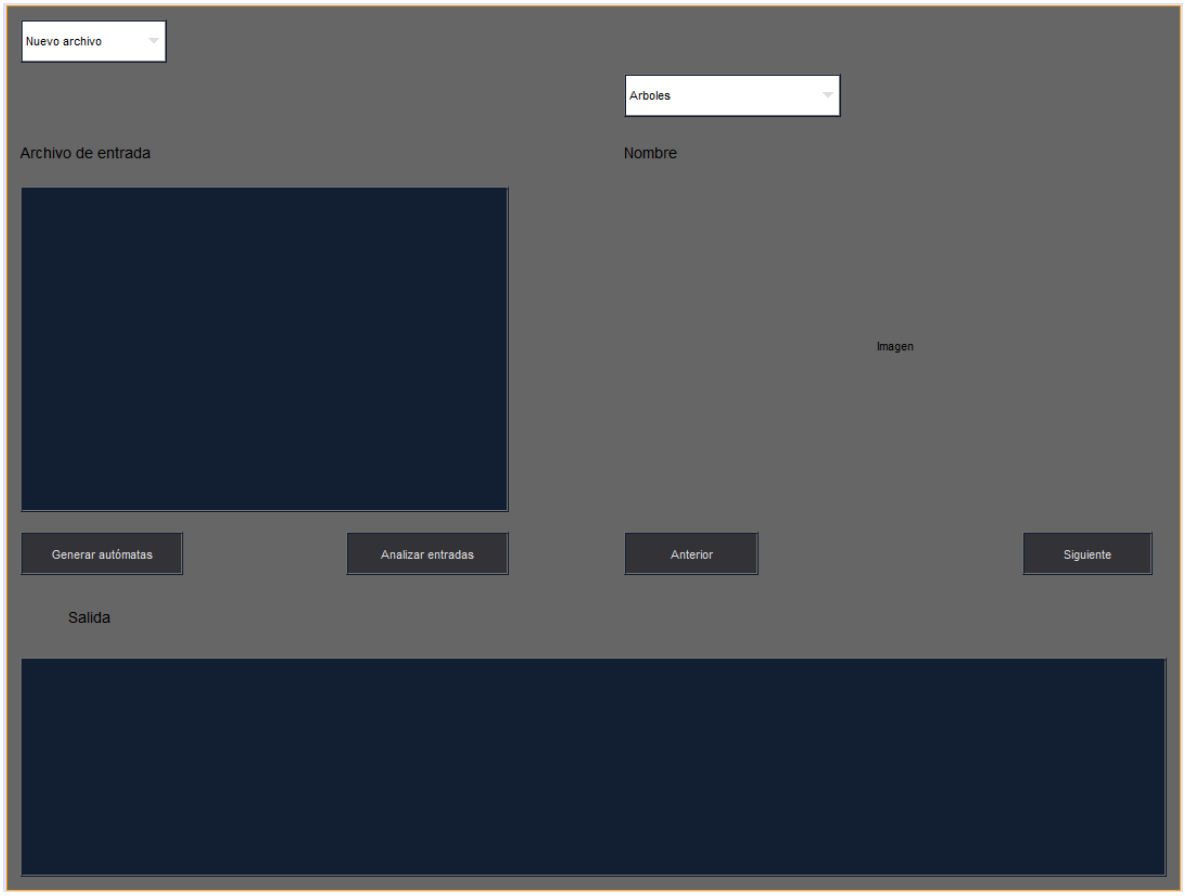


```
start with inicio;

inicio ::=
    LLAVELEFT expresion CAMBIOSECCION expresion2 LLAVERIGHT
    |LLAVELEFT expresion CAMBIOSECCION expresion2 LLAVERIGHT inicio
    |comentario inicio
    |comentario
    |SALTODELINEA inicio
    |SALTODELINEA
;

expresion ::=
    conjunto PUNTOYCOMA expresion
    |conjunto PUNTOYCOMA
    |regulares PUNTOYCOMA expresion
    |regulares PUNTOYCOMA
    |comentario expresion
    |comentario
    |SALTODELINEA expresion
    |SALTODELINEA
    |error PUNTOYCOMA
;

expresion2 ::=
    evaluar PUNTOYCOMA expresion2
    |evaluar PUNTOYCOMA
    |comentario expresion2
    |comentario
    |SALTODELINEA expresion2
    |SALTODELINEA
    |error PUNTOYCOMA
;

comentario ::=
    COMMENT //:a {: Proyectol_Compi.Menu.elementos.insertar(a); :}System.out.println("Comentario: "+a);
;

conjunto ::=
     CONJ DOSPUNTOS IDENTIFICADOR:a GUION MAYOR AZMINUS:b ONDULADO AZMINUS:c  {: String [] conjunto = {a, b, c}; Proyectol_Compi.Menu.conjuntos.add(conjunto); :}
    |CONJ DOSPUNTOS IDENTIFICADOR:a GUION MAYOR AZMAYUS:b ONDULADO AZMAYUS:c  {: String [] conjunto = {a, b, c}; Proyectol_Compi.Menu.conjuntos.add(conjunto); :}
    |CONJ DOSPUNTOS IDENTIFICADOR:a GUION MAYOR NUMERO:b ONDULADO NUMERO:c  {: String [] conjunto = {a, b, c}; Proyectol_Compi.Menu.conjuntos.add(conjunto); :}
    |CONJ DOSPUNTOS IDENTIFICADOR:a GUION MAYOR NOTACIONCOMAS:b  {: String [] conjunto = {a, b, ""}; Proyectol_Compi.Menu.conjuntos.add(conjunto); :}
    |CONJ DOSPUNTOS IDENTIFICADOR:a GUION MAYOR SIMBOLO:b ONDULADO SIMBOLO:c  {: String [] conjunto = {a, b, c}; Proyectol_Compi.Menu.conjuntos.add(conjunto); :}
;

regulares ::=
    IDENTIFICADOR:a GUION MAYOR polaca:b  {:NodoArbol root = new NodoArbol(".", b, new NodoArbol("#", null, null));
                                           ArbolBinario tree = new ArbolBinario(a, root);
```

Luego pasamos a la interfaz gráfica:

Interfaz gráfica:

La parte visual en un programa es muy importante, ya que por la vista se agrada al usuario, y esto sumado con la funcionalidad dan paso a que el usuario quede satisfecho.



Construcción del árbol binario

La clase árbol binario es la clase principal de nuestro programa, ya que con esta estructura se manejará toda la funcionalidad del análisis.

```
18  public class ArbolBinario {
19
20      private String nombre;
21      private NodoArbol root;
22      private int contadorNombreNodos;
23      private int contadorEstados = 1;
24
        private ArrayList<String> estadosAceptacion = new ArrayList();
        private ArrayList<String[]> siguientes = new ArrayList();
        private ArrayList<NodoEstado> estados = new ArrayList();
        private ArrayList<String[]> terminales = new ArrayList();
        private ArrayList<String[]> terminalesNoRepetidos = new ArrayList();
30
        private ArrayList<String[]> temporal = new ArrayList();
32
33      //Estado, entrada, transicion
        private ArrayList<String[]> transiciones = new ArrayList();
35      private String nameEstado = "";
36
37      //Constructor
38      public ArbolBinario(String nombre, NodoArbol root) {
39          this.nombre = nombre;
40          this.root = root;
41          this.contadorNombreNodos = 1;
42      }
```

Método para mostrar el árbol binario

Este método recibe un árbol binario y genera con código de graphviz una estructura en donde se guarda la expresión regular en notación polaca leída en el archivo de entrada.

```
public void generarGrafo(NodoArbol root) {
    String cadena = "\n\ndigraph G {\nnode[shape=box]\n";
    cadena += "label=\"" + nombre + "\";\n"; //Para el nombre del arbol
    cadena += "fontsize=50;\n"; //Para el nombre del arbol
    Queue<Object> cola = new LinkedList();
    NodoArbol au[] = new NodoArbol[2];
    au[0] = null;
    au[1] = root;
    cola.add(au);
    String N = "Nodo";
    while (cola.isEmpty() != true) {
        NodoArbol nodo[] = (NodoArbol[]) cola.remove();
        String label = "Simbolo: " + nodo[1].getValor() + "\n";
        label += "Primeros: " + nodo[1].getPrimeros() + "\n";
        label += "Ultimos: " + nodo[1].getUltimos() + "\n";
        label += "Anulable: " + nodo[1].isAnulable();
        if (nodo[0] != null) {

            cadena += N + nodo[1].hashCode() + "[label=\"" + label + "\"];\n";
            cadena += N + nodo[0].hashCode() + "->" + N + nodo[1].hashCode() + "\n";
        } else {

            cadena += N + nodo[1].hashCode() + "[label=\"" + label + "\"];\n";
        }
        if (nodo[1].getIzquierda() != null) {
            NodoArbol aux[] = new NodoArbol[2];
            aux[0] = nodo[1];
            aux[1] = nodo[1].getIzquierda();
            cola.add(aux);
        }
        if (nodo[1].getDerecha() != null) {
            NodoArbol aux2[] = new NodoArbol[2];
            aux2[0] = nodo[1];
            aux2[1] = nodo[1].getDerecha();
            cola.add(aux2);
        }
    }
    cadena += "}";
    generarGraphviz(cadena, "Reportes/Arboles_201901429/");
    //System.out.println("Cadena: " + nombre + " " + cadena);

}
```

Método para generar la imagen PNG con graphviz

El siguiente método recibe una cadena que contiene el código graphviz de una estructura y genera un archivo en formato png.

```java
    private void generarGraphviz(String cadena, String carpeta) {
        String nombreGrafo = carpeta + "grafo" + String.valueOf(Proyectol_Compi.Menu.contadorGrafosArboles);
        String path = carpeta + "grafo" + String.valueOf(Proyectol_Compi.Menu.contadorGrafosArboles) + ".png";
        Proyectol_Compi.Menu.contadorGrafosArboles++;
        FileWriter fichero = null;
        PrintWriter escritor;
        try {
            fichero = new FileWriter(nombreGrafo + ".dot");
            escritor = new PrintWriter(fichero);
            escritor.print(cadena);
        } catch (Exception e) {
            System.err.println("Error al escribir el archivvo");
        } finally {
            try {
                if (null != fichero) {
                    fichero.close();
                }
            } catch (Exception e2) {
                System.err.println("Error al cerrar el archivo");
            }
        }
        try {
            Runtime rt = Runtime.getRuntime();
            rt.exec("dot -Tpng -o " + path + " " + nombreGrafo + ".dot");
        } catch (Exception ex) {
            System.err.println("Error al generar la imagen");
        }
    }
```

Método principal para aplicar el método del árbol para expresiones regulares

En el siguiente método se aplican todos los pasos para realizar el análisis del método del arbol

```java
public void inicio() {

    nombrarNodos(this.root);            //Asigna los nombres a los nodos
    anulabilidad(this.root);            //Asigna la anulabilidad
    primerosNodoHoja(this.root);        //Primeras posiciones de los nodos hoja
    ultimosNodoHoja(this.root);         //Ultimas posiciones de los nodos hoja
    primerosYultimos(this.root);        //Calcula los primeros y los ulltimos de los nodos no hoja
    calcularSiguientes(this.root);      //Calcula los siguientes de todos los nodos

    declararTerminales(this.root);      //Para guardar todos los nodos hoja existentes

    tablaSiguientes(this.root);         //Creacion de la tabla de siguientes
    quitarRepetidosDeSiguientes(this.root); //Quita los repetidos de la tabla de siguientes y unifica los conjuntos de los repetidos

    crearPrimerEstado();                //Crea el estado S0
    crearEstados();                     //Crea todos los demás estados a partir del S1

    generarGrafo(this.root);            //Genera el árbol
    generarSiguientes();                //Genera la tabla de siguientes
    generarTransiciones();              //Genera la tabla de transiciones

    String simboloAceptacion = String.valueOf(this.root.getDerecha().getNumero());
    estadosAceptacion(simboloAceptacion);
    generarAFD();                       //Genera el AFD
}
```