Archivo LEX

```
/*Compiladores 2
  2007
*/

//Importa las clases que utilza para devolver los simbolos
import java_cup.runtime.Symbol;

class cadena
{
		public String cad;

		public cadena(String cad_)
		{
			cad=new String(cad_);
		}

		public cadena(String cad1_,String cad2_)
		{
			cad=new String(cad1_+","+cad2_);
		}

}


/*Para que sea compatible con cup
  %cup

  Hace la clase publica
  %public

  Agrega todos los caracteres, para que este disponible la ñ
  %full

  Para utilizar el atributo line
  %line

  Para utilizar el atributo char
  %char

  Coloca el case insensitive, es decir no hace diferencia entre mayusculas y minusculas
  %ignorecase

  Esto genera el simbolo de final de archivo
  %eofval
*/

/*los nombre que aparecen despues de sym. son los nombre que se le dan a los TERMINALES,
  despues se debe de especificar el tipo que se devolvera, regularmente String, pero puede
  ser cualquier otro tipo y despues convertirlo
*/

/*Nota:
  Jlex no acepta comentarios abajo de este simbolo de doble porcentaje.
```

```
*/

%%

%{
 //Declaro variables y código que será copiado al escaner
 public static int linea=1;
 public static int pos=0;

 /*Nota:
   Aca si puedo comentar
 */

%}

%cup
%public
%full
%line
%char
%ignorecase
%eofval{
 return new Symbol(sym.EOF,new String("Fin de Archivo"));
%eofval}

%%
","                             { /*otro tipo de acciones por ejemplo aumentar la columna o fila*/
                                  return new Symbol(sym.COMA, new String(yytext())); }

[0-9]+                          { return new Symbol(sym.NUMBER, new Integer(yytext())); }
[a-zA-Z]+       { return new Symbol(sym.id, new String(yytext())); }



[ \t\r\n\f]     { /* Ignora espacios en blanco */ }
"."                             {return new Symbol(sym.PUNTO);
                                 /*al estar encerrado entre comillas devuelve el token PUNTO cuando lea
                                   un punto de la entrada. Notese la diferencia con la siguiente instruccion
                                   donde el punto aparece sin comillas. Cuando se coloca un punto sin comillas
                                   junto a sus acciones estas acciones serán ejecutadas cada vez que el analizador
                                   lexico encuentre algun lexema que no concuerde con cualquiera de los
                                   componentes lexicos especificados. Por ejemplo, al encontrar un simbolo
                                   como la @ estre imprimira en la shell "Caracter Ilegal: @". la función
                                   yytext() devuelve el ultimo lexema levantado por el analizador lexico.
                                 */
                                }
.                               { System.out.println("Caracter Ilegal: "+yytext()); }
```

Archivo Cup

```
 /*Compiladores 2
  2007
*/

/*Aqui se incluyen todos los paquetes que se vayan a utilizar dentro del proyecto
  Por ejemplo se utilizará la clase ArrayList se debe agregar la linea
  imports Java.util.ArrayList
*/

import java.io.*;

//Clase incluida para las instrucciones de CUP
import java_cup.runtime.*;

/*Codigo que se copia a la clase parser que genera, para poder manejarlo dentro de las acciones semánticas
  este código se copia exactamente igual, no revisa errores
*/
parser code
{:
    public static void main(String args[]) throws Exception{
      new parser(new Yylex(new FileInputStream(args[0]))).parse();
    } // fin procedimiento

    public void syntax_error(Symbol s){
                //aca se define el manejo del error que provee CUP
                report_error("Error de sintaxis Linea:"+Integer.toString(Yylex.linea+1)+"
Columna:"+Integer.toString(Yylex.pos+1)+" En \""+s.value+"\"",null);
    } // fin procedimiento
:}

//Sección donde se pueden declarar variables, constantes, etc
action code
{:              //mis variables
                int integer = 1;
:}

//Declaración de Terminales y los tipos que usaran


terminal String id,COMA;
terminal Integer NUMBER;
terminal PUNTO;

//Declaracion de no terminales y los tipos que usarán

non terminal cadena expr;
non terminal cadena expr_list, expr_part;

/*Gramática, el lado izquierdo produce lo del lado derecho
  Las acciones semánticas se colocan entre llaves dos puntos, dos puntos llaves {: :}
  La expresion :identificador, nos permite adquirir un alias del no terminal o terminal
  para poder utilizarlo como variable, del tipo que se le asignó, en las acciones semánticas
*/
```

```
expr ::= expr_list:a {:System.out.println(a.cad);:};

expr_list::= expr_list:a COMA expr_part:b {:RESULT=new cadena(a.cad,b.cad);:}
                    | expr_part:a {:RESULT=new cadena(a.cad);:}
                    ;

expr_part ::= id:a {:RESULT=new cadena(a);:};
```

ENTRADA

alfa, beta, gama