```
------------------------------
--     MAIN.JAVA            --
------------------------------
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.lang.String;

/* Classe principal do programa, responsavel pela abertura do arquivo
 * de entrada e invocacao da classe Lexer que faz a leitura dos tokens.
 * E' responsavel tambem pela criacao dos arquivos de saida.
 */

public class Main
{

  public static void main(String[] args)
  {
    String sArquivoEntrada;
    try
    {
      sArquivoEntrada = args[0];
      // Aqui e' iniciada a leitura do arquivo de entrada
      Lexer lex = new Lexer(new FileReader(sArquivoEntrada));
      // Aqui e' invocada a funcao principal funcao do programa, que faz o
      // reconhecimento dos tokens baseado nas definicoes da gramatica.
      Token tToken = lex.yylex();
      // Aqui e' gerado o arquivo de saida que contem o resultado da analise
      // lexica.
      FileWriter writer = new FileWriter(new File("saida_" + sArquivoEntrada));
      PrintWriter saida = new PrintWriter(writer,true);

      if (tToken != null)
      {
        saida.println(String.format("%20s %10s %10s %10s", "Token", "Lexema", "C
oluna", "Linha"));
      }
      while (tToken != null)
      {
        saida.println(tToken.createString());
        tToken = lex.yylex();
      }

    }
    catch (FileNotFoundException e)
    {
      System.out.println("O arquivo não foi encontrado");
    }
    catch (IOException e)
    {
      System.out.println("Erro de IO");
    }

  }

}



------------------------------
--     TOKEN.JAVA  X        --
------------------------------
// Classe que contem a definicao da estrutura dos tokens

class Token
{
```



```
  private String sName;
  private String sText;
  private int    iLine;
  private int    iColumn;

  // Método construtor da classe
  Token (String name, String text, int line, int column)
  {
    sName   = name;
    sText   = text;
    iLine   = line;
    iColumn = column;
  }

  // Define o formato como os atributos serão impressos na saída
  public String createString()
  {
    return String.format("%20s %10s %10d %10d", sName, sText, iLine, iColumn);
  }
}




------------------------------
--       LEXER.JAVA         --
------------------------------
/* The following code was generated by JFlex 1.4.3 on 5/6/09 6:39 PM */

/**
 * This class is a scanner generated by
 * <a href="http://www.jflex.de/">JFlex</a> 1.4.3
 * on 5/6/09 6:39 PM from the specification file
 * <tt>tp3.flex</tt>
 */
class Lexer {

  /** This character denotes the end of file */
  public static final int YYEOF = -1;

  /** initial size of the lookahead buffer */
  private static final int ZZ_BUFFERSIZE = 16384;

  /** lexical states */
  public static final int YYINITIAL = 0;

  /**
   * ZZ_LEXSTATE[l] is the state in the DFA for the lexical state l
   * ZZ_LEXSTATE[l+1] is the state in the DFA for the lexical state l
   *                  at the beginning of a line
   * l is of the form l = 2*k, k a non negative integer
   */
  private static final int ZZ_LEXSTATE[] = {
     0, 0
  };

  /**
   * Translates characters to character classes
   */
  private static final String ZZ_CMAP_PACKED =
    "\10\0\2\31\1\32\2\0\1\7\22\0\1\31\1\13\5\0\1\6"+
    "\1\51\1\52\1\22\1\17\1\50\1\20\1\5\1\23\12\2\1\47"+
    "\1\46\1\11\1\10\1\12\2\0\4\1\1\4\25\1\1\53\1\0"+
    "\1\54\3\0\1\24\1\40\1\27\1\25\1\36\1\44\1\34\1\42"+
    "\1\41\2\1\1\37\1\35\1\14\1\15\1\33\1\1\1\21\1\30"+
    "\1\16\1\26\1\1\1\45\1\1\1\43\1\1\1\0\1\3\uff83\0";

  /**
   * Translates characters to character classes
```

```
 */
private static final char [] ZZ_CMAP = zzUnpackCMap(ZZ_CMAP_PACKED);

/**
 * Translates DFA states to action switch labels.
 */
private static final int [] ZZ_ACTION = zzUnpackAction();

private static final String ZZ_ACTION_PACKED_0 =
  "\1\0\1\1\1\2\1\3\1\1\1\4\1\5\1\6"+
  "\1\7\1\1\3\2\1\10\1\11\1\2\1\12\1\13"+
  "\5\2\1\4\10\2\1\14\1\15\1\16\1\17\1\20"+
  "\1\21\1\22\1\0\1\23\1\0\1\4\1\24\1\25"+
  "\1\26\2\2\1\27\1\30\3\2\1\0\2\2\1\31"+
  "\14\2\1\32\3\2\1\33\1\23\1\0\1\34\1\35"+
  "\6\2\1\36\7\2\1\37\11\2\1\40\1\41\1\2"+
  "\1\42\1\43\4\2\1\44\1\45\2\2\1\46\1\47"+
  "\12\2\1\50\1\2\1\51\2\2\1\52\1\2\1\53"+
  "\1\2\1\54\1\55\1\56\1\2\1\57\1\60\6\2"+
  "\1\61\1\2\1\62\1\63\1\64\2\2\1\65\1\66";

private static int [] zzUnpackAction() {
  int [] result = new int[157];
  int offset = 0;
  offset = zzUnpackAction(ZZ_ACTION_PACKED_0, offset, result);
  return result;
}

private static int zzUnpackAction(String packed, int offset, int [] result) {
  int i = 0;       /* index in packed string  */
  int j = offset;  /* index in unpacked array */
  int l = packed.length();
  while (i < l) {
    int count = packed.charAt(i++);
    int value = packed.charAt(i++);
    do result[j++] = value; while (--count > 0);
  }
  return j;
}


/**
 * Translates a state to a row index in the transition table
 */
private static final int [] ZZ_ROWMAP = zzUnpackRowMap();

private static final String ZZ_ROWMAP_PACKED_0 =
  "\0\0\0\55\0\132\0\207\0\264\0\341\0\55\0\u010e"+
  "\0\u013b\0\u0168\0\u0195\0\u01c2\0\u01ef\0\55\0\55\0\u021c"+
  "\0\55\0\u0249\0\u0276\0\u02a3\0\u02d0\0\u02fd\0\u032a\0\u0357"+
  "\0\u0384\0\u03b1\0\u03de\0\u040b\0\u0438\0\u0465\0\u0492\0\u04bf"+
  "\0\55\0\u04ec\0\55\0\55\0\55\0\55\0\55\0\u0519"+
  "\0\u0546\0\u0573\0\55\0\55\0\55\0\55\0\u05a0\0\u05cd"+
  "\0\132\0\132\0\u05fa\0\u0627\0\u0654\0\u0681\0\u06ae\0\u06db"+
  "\0\132\0\u0708\0\u0735\0\u0762\0\u078f\0\u07bc\0\u07e9\0\u0816"+
  "\0\u0843\0\u0870\0\u089d\0\u08ca\0\u08f7\0\132\0\u0924\0\u0951"+
  "\0\u097e\0\55\0\u09ab\0\u09ab\0\55\0\132\0\u09d8\0\u0a05"+
  "\0\u0a32\0\u0a5f\0\u0a8c\0\u0ab9\0\132\0\u0ae6\0\u0b13\0\u0b40"+
  "\0\u0b6d\0\u0b9a\0\u0bc7\0\u0bf4\0\132\0\u0c21\0\u0c4e\0\u0c7b"+
  "\0\u0ca8\0\u0cd5\0\u0d02\0\u0d2f\0\u0d5c\0\u0d89\0\132\0\132"+
  "\0\u0db6\0\132\0\132\0\u0de3\0\u0e10\0\u0e3d\0\u0e6a\0\132"+
  "\0\132\0\u0e97\0\u0ec4\0\132\0\132\0\u0ef1\0\u0f1e\0\u0f4b"+
  "\0\u0f78\0\u0fa5\0\u0fd2\0\u0fff\0\u102c\0\u1059\0\u1086\0\132"+
  "\0\u10b3\0\132\0\u10e0\0\u110d\0\132\0\u113a\0\132\0\u1167"+
  "\0\132\0\132\0\132\0\u1194\0\132\0\132\0\u11c1\0\u11ee"+
  "\0\u121b\0\u1248\0\u1275\0\u12a2\0\132\0\u12cf\0\132\0\132"+
  "\0\132\0\u12fc\0\u1329\0\132\0\132";

private static int [] zzUnpackRowMap() {
```



```
  int [] result = new int[157];
  int offset = 0;
  offset = zzUnpackRowMap(ZZ_ROWMAP_PACKED_0, offset, result);
  return result;
}

private static int zzUnpackRowMap(String packed, int offset, int [] result) {
  int i = 0;  /* index in packed string  */
  int j = offset;  /* index in unpacked array */
  int l = packed.length();
  while (i < l) {
    int high = packed.charAt(i++) << 16;
    result[j++] = high | packed.charAt(i++);
  }
  return j;
}

/**
 * The transition table of the DFA
 */
private static final int [] ZZ_TRANS = zzUnpackTrans();

private static final String ZZ_TRANS_PACKED_0 =
  "\1\2\1\3\1\4\1\2\1\3\1\2\1\5\1\6"+
  "\1\7\1\10\1\11\1\12\1\13\1\14\1\15\1\16"+
  "\1\17\1\20\1\21\1\22\1\23\1\24\1\25\1\26"+
  "\1\27\2\30\1\31\1\32\1\27\1\33\1\34\1\35"+
  "\1\36\2\3\1\37\1\40\1\41\1\42\1\43\1\44"+
  "\1\45\1\46\1\47\56\0\2\3\1\0\1\3\7\0"+
  "\3\3\2\0\1\3\2\0\5\3\2\0\13\3\11\0"+
  "\1\4\1\0\1\50\1\51\47\0\7\52\1\0\22\52"+
  "\1\0\22\52\32\0\1\53\32\0\1\54\54\0\1\55"+
  "\54\0\1\56\45\0\2\3\1\0\1\3\7\0\1\27"+
  "\1\57\1\3\2\0\1\3\2\0\2\3\1\27\1\60"+
  "\1\27\2\0\2\3\1\27\3\3\1\27\4\3\10\0"+
  "\2\3\1\0\1\3\7\0\3\3\2\0\1\61\2\0"+
  "\5\3\2\0\11\3\1\62\1\3\10\0\2\3\1\0"+
  "\1\3\7\0\3\3\2\0\1\63\2\0\5\3\2\0"+
  "\7\3\1\64\3\3\10\0\2\3\1\0\1\3\7\0"+
  "\3\3\2\0\1\3\2\0\5\3\2\0\3\3\1\65"+
  "\7\3\32\0\1\66\32\0\2\3\1\0\1\3\7\0"+
  "\1\67\2\3\2\0\1\70\2\0\5\3\2\0\13\3"+
  "\10\0\2\3\1\0\1\3\7\0\1\3\1\71\1\3"+
  "\2\0\1\3\2\0\5\3\2\0\3\3\1\72\7\3"+
  "\10\0\2\3\1\0\1\3\7\0\1\73\2\3\2\0"+
  "\1\3\2\0\2\3\1\27\1\60\1\27\2\0\2\3"+
  "\1\27\3\3\1\27\4\3\10\0\2\3\1\0\1\3"+
  "\7\0\3\3\2\0\1\3\2\0\1\74\4\3\2\0"+
  "\7\3\1\75\3\3\10\0\2\3\1\0\1\3\7\0"+
  "\1\27\2\3\2\0\1\3\2\0\2\3\1\27\1\60"+
  "\1\27\2\0\2\3\1\27\3\3\1\27\4\3\40\0"+
  "\2\30\23\0\2\3\1\0\1\3\7\0\3\3\2\0"+
  "\1\76\2\0\5\3\2\0\13\3\10\0\2\3\1\0"+
  "\1\3\7\0\1\3\1\77\1\3\2\0\1\3\2\0"+
  "\5\3\2\0\13\3\10\0\2\3\1\0\1\3\7\0"+
  "\1\100\2\3\2\0\1\3\2\0\5\3\2\0\4\3"+
  "\1\101\6\3\10\0\2\3\1\0\1\3\7\0\3\3"+
  "\2\0\1\3\2\0\1\102\4\3\2\0\13\3\10\0"+
  "\2\3\1\0\1\3\7\0\1\3\1\103\1\3\2\0"+
  "\1\3\2\0\5\3\2\0\3\3\1\104\7\3\10\0"+
  "\2\3\1\0\1\3\7\0\1\105\2\3\2\0\1\3"+
  "\2\0\2\3\1\27\1\60\1\27\2\0\2\3\1\27"+
  "\3\3\1\27\2\3\1\106\1\3\10\0\2\3\1\0"+
  "\1\3\7\0\3\3\2\0\1\3\2\0\1\107\4\3"+
  "\2\0\13\3\10\0\2\3\1\0\1\3\7\0\3\3"+
  "\2\0\1\110\2\0\5\3\2\0\7\3\1\111\3\3"+
  "\17\0\1\112\46\0\1\113\1\114\13\0\2\114\36\0"+
  "\1\51\1\0\1\50\56\0\1\115\47\0\2\3\1\0"+
  "\1\3\7\0\2\3\1\116\2\0\1\3\2\0\5\3"+
```

```
    "\2\0\13\3\10\0\2\3\1\0\1\3\7\0\1\3"+
    "\1\117\1\3\2\0\1\3\2\0\5\3\2\0\13\3"+
    "\10\0\2\3\1\0\1\3\7\0\3\3\2\0\1\3"+
    "\2\0\2\3\1\120\2\3\2\0\13\3\10\0\2\3"+
    "\1\0\1\3\7\0\3\3\2\0\1\3\2\0\5\3"+
    "\2\0\3\3\1\121\7\3\10\0\2\3\1\0\1\3"+
    "\7\0\2\3\1\122\2\0\1\3\2\0\1\123\4\3"+
    "\2\0\1\124\12\3\7\0\7\66\1\6\22\66\1\53"+
    "\22\66\1\0\2\3\1\0\1\3\7\0\3\3\2\0"+
    "\1\3\2\0\1\3\1\125\3\3\2\0\13\3\10\0"+
    "\2\3\1\0\1\3\7\0\3\3\2\0\1\126\2\0"+
    "\5\3\2\0\13\3\10\0\2\3\1\0\1\3\7\0"+
    "\3\3\2\0\1\3\2\0\3\3\1\127\1\3\2\0"+
    "\13\3\10\0\2\3\1\0\1\3\7\0\1\27\1\3"+
    "\1\130\2\0\1\3\2\0\2\3\1\27\1\60\1\27"+
    "\2\0\2\3\1\27\3\3\1\27\4\3\10\0\2\3"+
    "\1\0\1\3\7\0\3\3\2\0\1\3\2\0\4\3"+
    "\1\131\2\0\13\3\10\0\2\3\1\0\1\3\7\0"+
    "\3\3\2\0\1\3\2\0\1\132\4\3\2\0\13\3"+
    "\10\0\2\3\1\0\1\3\7\0\1\3\1\133\1\3"+
    "\2\0\1\3\2\0\5\3\2\0\13\3\10\0\2\3"+
    "\1\0\1\3\7\0\2\3\1\134\2\0\1\3\2\0"+
    "\5\3\2\0\13\3\10\0\2\3\1\0\1\3\7\0"+
    "\3\3\2\0\1\3\2\0\1\3\1\135\3\3\2\0"+
    "\13\3\10\0\2\3\1\0\1\3\7\0\3\3\2\0"+
    "\1\3\2\0\4\3\1\136\2\0\13\3\10\0\2\3"+
    "\1\0\1\3\7\0\3\3\2\0\1\3\2\0\5\3"+
    "\2\0\5\3\1\137\5\3\10\0\2\3\1\0\1\3"+
    "\7\0\1\3\1\140\1\3\2\0\1\3\2\0\5\3"+
    "\2\0\13\3\10\0\2\3\1\0\1\3\7\0\3\3"+
    "\2\0\1\3\2\0\1\3\1\141\11\3"+
    "\10\0\2\3\1\0\1\3\7\0\1\27\1\3\1\142"+
    "\2\0\1\3\2\0\2\3\1\27\1\60\1\27\2\0"+
    "\2\3\1\27\3\3\1\27\4\3\10\0\2\3\1\0"+
    "\1\3\7\0\3\3\2\0\1\3\2\0\5\3\2\0"+
    "\4\3\1\143\6\3\10\0\2\3\1\0\1\3\7\0"+
    "\3\3\2\0\1\3\2\0\5\3\2\0\6\3\1\144"+
    "\4\3\10\0\2\3\1\0\1\3\7\0\3\3\2\0"+
    "\1\3\2\0\5\3\2\0\6\3\1\145\4\3\11\0"+
    "\1\113\53\0\2\3\1\0\1\3\7\0\1\146\2\3"+
    "\2\0\1\3\2\0\5\3\2\0\13\3\10\0\2\3"+
    "\1\0\1\3\7\0\3\3\2\0\1\3\2\0\5\3"+
    "\2\0\3\3\1\147\7\3\10\0\2\3\1\0\1\3"+
    "\7\0\1\150\2\3\2\0\1\3\2\0\5\3\2\0"+
    "\13\3\10\0\2\3\1\0\1\3\7\0\3\3\2\0"+
    "\1\3\2\0\2\3\1\151\2\3\2\0\13\3\10\0"+
    "\2\3\1\0\1\3\7\0\3\3\2\0\1\3\2\0"+
    "\1\3\1\152\3\3\2\0\4\3\1\153\6\3\10\0"+
    "\2\3\1\0\1\3\7\0\3\3\2\0\1\3\2\0"+
    "\5\3\2\0\3\3\1\154\7\3\10\0\2\3\1\0"+
    "\1\3\7\0\3\3\2\0\1\3\2\0\1\155\4\3"+
    "\2\0\13\3\10\0\2\3\1\0\1\3\7\0\3\3"+
    "\2\0\1\3\2\0\5\3\2\0\4\3\1\156\6\3"+
    "\10\0\2\3\1\0\1\3\7\0\3\3\2\0\1\3"+
    "\2\0\5\3\2\0\6\3\1\157\4\3\10\0\2\3"+
    "\1\0\1\3\7\0\3\3\2\0\1\3\2\0\5\3"+
    "\2\0\3\3\1\160\7\3\10\0\2\3\1\0\1\3"+
    "\7\0\3\3\2\0\1\161\2\0\5\3\2\0\13\3"+
    "\10\0\2\3\1\0\1\3\7\0\3\3\2\0\1\3"+
    "\2\0\3\3\1\162\1\3\2\0\1\3\1\163\11\3"+
    "\10\0\2\3\1\0\1\3\7\0\1\3\1\164\1\3"+
    "\2\0\1\3\2\0\5\3\2\0\13\3\10\0\2\3"+
    "\1\0\1\3\7\0\3\3\2\0\1\3\2\0\5\3"+
    "\2\0\3\3\1\165\7\3\10\0\2\3\1\0\1\3"+
    "\7\0\3\3\2\0\1\3\2\0\5\3\2\0\3\3"+
    "\1\166\7\3\10\0\2\3\1\0\1\3\7\0\3\3"+
    "\2\0\1\3\2\0\5\3\2\0\4\3\1\167\6\3"+
    "\10\0\2\3\1\0\1\3\7\0\3\3\2\0\1\3"+
    "\2\0\5\3\2\0\6\3\1\170\4\3\10\0\2\3"+
```



```
    "\1\0\1\3\7\0\3\3\2\0\1\3\2\0\5\3"+
    "\2\0\3\3\1\171\7\3\10\0\2\3\1\0\1\3"+
    "\7\0\3\3\2\0\1\3\2\0\4\3\1\172\2\0"+
    "\13\3\10\0\2\3\1\0\1\3\7\0\2\3\1\173"+
    "\2\0\1\3\2\0\5\3\2\0\13\3\10\0\2\3"+
    "\1\0\1\3\7\0\3\3\2\0\1\3\2\0\5\3"+
    "\2\0\4\3\1\174\6\3\10\0\2\3\1\0\1\3"+
    "\7\0\3\3\2\0\1\3\2\0\4\3\1\175\2\0"+
    "\13\3\10\0\2\3\1\0\1\3\7\0\3\3\2\0"+
    "\1\176\2\0\5\3\2\0\13\3\10\0\2\3\1\0"+
    "\1\3\7\0\3\3\2\0\1\3\2\0\1\177\4\3"+
    "\2\0\13\3\10\0\2\3\1\0\1\3\7\0\3\3"+
    "\2\0\1\3\2\0\5\3\2\0\10\3\1\200\2\3"+
    "\10\0\2\3\1\0\1\3\7\0\3\3\2\0\1\3"+
    "\2\0\1\201\4\3\2\0\13\3\10\0\2\3\1\0"+
    "\1\3\7\0\3\3\2\0\1\3\2\0\5\3\2\0"+
    "\4\3\1\202\6\3\10\0\2\3\1\0\1\3\7\0"+
    "\3\3\2\0\1\3\2\0\5\3\2\0\3\3\1\203"+
    "\7\3\10\0\2\3\1\0\1\3\7\0\3\3\2\0"+
    "\1\204\2\0\5\3\2\0\13\3\10\0\2\3\1\0"+
    "\1\3\7\0\3\3\2\0\1\3\2\0\5\3\2\0"+
    "\4\3\1\205\6\3\10\0\2\3\1\0\1\3\7\0"+
    "\3\3\2\0\1\3\2\0\5\3\2\0\3\3\1\206"+
    "\7\3\10\0\2\3\1\0\1\3\7\0\1\207\2\3"+
    "\2\0\1\3\2\0\5\3\2\0\13\3\10\0\2\3"+
    "\1\0\1\3\7\0\3\3\2\0\1\3\2\0\5\3"+
    "\2\0\1\3\1\210\11\3\10\0\2\3\1\0\1\3"+
    "\7\0\3\3\2\0\1\3\2\0\5\3\2\0\3\3"+
    "\1\211\7\3\10\0\2\3\1\0\1\3\7\0\3\3"+
    "\2\0\1\3\2\0\5\3\2\0\3\3\1\212\7\3"+
    "\10\0\2\3\1\0\1\3\7\0\3\3\2\0\1\3"+
    "\2\0\5\3\2\0\3\3\1\213\7\3\10\0\2\3"+
    "\1\0\1\3\7\0\2\3\1\214\2\0\1\3\2\0"+
    "\5\3\2\0\13\3\10\0\2\3\1\0\1\3\7\0"+
    "\1\215\2\3\2\0\1\3\2\0\5\3\2\0\13\3"+
    "\10\0\2\3\1\0\1\3\7\0\2\3\1\216\2\0"+
    "\1\3\2\0\5\3\2\0\13\3\10\0\2\3\1\0"+
    "\1\3\7\0\3\3\2\0\1\217\2\0\5\3\2\0"+
    "\13\3\10\0\2\3\1\0\1\3\7\0\3\3\2\0"+
    "\1\3\2\0\1\3\1\220\3\3\2\0\13\3\10\0"+
    "\2\3\1\0\1\3\7\0\3\3\2\0\1\3\2\0"+
    "\1\221\4\3\2\0\13\3\10\0\2\3\1\0\1\3"+
    "\7\0\3\3\2\0\1\3\2\0\1\222\4\3\2\0"+
    "\13\3\10\0\2\3\1\0\1\3\7\0\3\3\2\0"+
    "\1\3\2\0\5\3\2\0\3\3\1\223\7\3\10\0"+
    "\2\3\1\0\1\3\7\0\3\3\2\0\1\3\2\0"+
    "\1\224\4\3\2\0\13\3\10\0\2\3\1\0\1\3"+
    "\7\0\3\3\2\0\1\3\2\0\5\3\2\0\3\3"+
    "\1\225\7\3\10\0\2\3\1\0\1\3\7\0\3\3"+
    "\2\0\1\3\2\0\2\3\1\226\2\3\2\0\13\3"+
    "\10\0\2\3\1\0\1\3\7\0\3\3\2\0\1\3"+
    "\2\0\5\3\2\0\2\3\1\227\10\3\10\0\2\3"+
    "\1\0\1\3\7\0\1\230\2\3\2\0\1\3\2\0"+
    "\5\3\2\0\13\3\10\0\2\3\1\0\1\3\7\0"+
    "\3\3\2\0\1\231\2\0\5\3\2\0\13\3\10\0"+
    "\2\3\1\0\1\3\7\0\1\232\2\3\2\0\1\3"+
    "\2\0\5\3\2\0\13\3\10\0\2\3\1\0\1\3"+
    "\7\0\3\3\2\0\1\233\2\0\5\3\2\0\13\3"+
    "\10\0\2\3\1\0\1\3\7\0\2\3\1\234\2\0"+
    "\1\3\2\0\5\3\2\0\13\3\10\0\2\3\1\0"+
    "\1\3\7\0\3\3\2\0\1\3\2\0\5\3\2\0"+
    "\3\3\1\235\7\3\7\0";

 private static int [] zzUnpackTrans() {
   int [] result = new int[4950];
   int offset = 0;
   offset = zzUnpackTrans(ZZ_TRANS_PACKED_0, offset, result);
   return result;
 }
```

```
  private static int zzUnpackTrans(String packed, int offset, int [] result) {
    int i = 0;       /* index in packed string  */
    int j = offset;  /* index in unpacked array */
    int l = packed.length();
    while (i < l) {
      int count = packed.charAt(i++);
      int value = packed.charAt(i++);
      value--;
      do result[j++] = value; while (--count > 0);
    }
    return j;
  }


  /* error codes */
  private static final int ZZ_UNKNOWN_ERROR = 0;
  private static final int ZZ_NO_MATCH = 1;
  private static final int ZZ_PUSHBACK_2BIG = 2;

  /* error messages for the codes above */
  private static final String ZZ_ERROR_MSG[] = {
    "Unkown internal scanner error",
    "Error: could not match input",
    "Error: pushback value was too large"
  };

  /**
   * ZZ_ATTRIBUTE[aState] contains the attributes of state <code>aState</code>
   */
  private static final int [] ZZ_ATTRIBUTE = zzUnpackAttribute();

  private static final String ZZ_ATTRIBUTE_PACKED_0 =
    "\1\0\1\11\4\1\1\11\6\1\2\11\1\1\1\11"+
    "\17\1\1\11\1\1\5\11\1\0\1\1\1\0\4\11"+
    "\7\1\1\0\23\1\1\11\1\1\1\0\1\11\120\1";

  private static int [] zzUnpackAttribute() {
    int [] result = new int[157];
    int offset = 0;
    offset = zzUnpackAttribute(ZZ_ATTRIBUTE_PACKED_0, offset, result);
    return result;
  }

  private static int zzUnpackAttribute(String packed, int offset, int [] result)
{
    int i = 0;       /* index in packed string  */
    int j = offset;  /* index in unpacked array */
    int l = packed.length();
    while (i < l) {
      int count = packed.charAt(i++);
      int value = packed.charAt(i++);
      do result[j++] = value; while (--count > 0);
    }
    return j;
  }

  /** the input device */
  private java.io.Reader zzReader;

  /** the current state of the DFA */
  private int zzState;

  /** the current lexical state */
  private int zzLexicalState = YYINITIAL;

  /** this buffer contains the current text to be matched and is
      the source of the yytext() string */
  private char zzBuffer[] = new char[ZZ_BUFFERSIZE];
```



```
  /** the textposition at the last accepting state */
  private int zzMarkedPos;

  /** the current text position in the buffer */
  private int zzCurrentPos;

  /** startRead marks the beginning of the yytext() string in the buffer */
  private int zzStartRead;

  /** endRead marks the last character in the buffer, that has been read
      from input */
  private int zzEndRead;

  /** number of newlines encountered up to the start of the matched text */
  private int yyline;

  /** the number of characters up to the start of the matched text */
  private int yychar;

  /**
   * the number of characters from the last newline up to the start of the
   * matched text
   */
  private int yycolumn;

  /**
   * zzAtBOL == true <=> the scanner is currently at the beginning of a line
   */
  private boolean zzAtBOL = true;

  /** zzAtEOF == true <=> the scanner is at the EOF */
  private boolean zzAtEOF;

  /** denotes if the user-EOF-code has already been executed */
  private boolean zzEOFDone;

  /* user code: */



  /**
   * Creates a new scanner
   * There is also a java.io.InputStream version of this constructor.
   *
   * @param   in  the java.io.Reader to read input from.
   */
  Lexer(java.io.Reader in) {
    this.zzReader = in;
  }

  /**
   * Creates a new scanner.
   * There is also java.io.Reader version of this constructor.
   *
   * @param   in  the java.io.Inputstream to read input from.
   */
  Lexer(java.io.InputStream in) {
    this(new java.io.InputStreamReader(in));
  }

  /**
   * Unpacks the compressed character translation table.
   *
   * @param packed   the packed character translation table
   * @return         the unpacked character translation table
   */
  private static char [] zzUnpackCMap(String packed) {
    char [] map = new char[0x10000];
```

```
    int i = 0;  /* index in packed string  */
    int j = 0;  /* index in unpacked array */
    while (i < 120) {
      int  count = packed.charAt(i++);
      char value = packed.charAt(i++);
      do map[j++] = value; while (--count > 0);
    }
    return map;
  }


  /**
   * Refills the input buffer.
   *
   * @return      <code>false</code>, iff there was new input.
   *
   * @exception   java.io.IOException  if any I/O-Error occurs
   */
  private boolean zzRefill() throws java.io.IOException {

    /* first: make room (if you can) */
    if (zzStartRead > 0) {
      System.arraycopy(zzBuffer, zzStartRead,
                       zzBuffer, 0,
                       zzEndRead-zzStartRead);

      /* translate stored positions */
      zzEndRead-= zzStartRead;
      zzCurrentPos-= zzStartRead;
      zzMarkedPos-= zzStartRead;
      zzStartRead = 0;
    }

    /* is the buffer big enough? */
    if (zzCurrentPos >= zzBuffer.length) {
      /* if not: blow it up */
      char newBuffer[] = new char[zzCurrentPos*2];
      System.arraycopy(zzBuffer, 0, newBuffer, 0, zzBuffer.length);
      zzBuffer = newBuffer;
    }

    /* finally: fill the buffer with new input */
    int numRead = zzReader.read(zzBuffer, zzEndRead,
                                            zzBuffer.length-zzEndRead);

    if (numRead > 0) {
      zzEndRead+= numRead;
      return false;
    }
    // unlikely but not impossible: read 0 characters, but not at end of stream

    if (numRead == 0) {
      int c = zzReader.read();
      if (c == -1) {
        return true;
      } else {
        zzBuffer[zzEndRead++] = (char) c;
        return false;
      }
    }

        // numRead < 0
    return true;
  }


  /**
   * Closes the input stream.
   */
```



```
  public final void yyclose() throws java.io.IOException {
    zzAtEOF = true;            /* indicate end of file */
    zzEndRead = zzStartRead;  /* invalidate buffer    */

    if (zzReader != null)
      zzReader.close();
  }


  /**
   * Resets the scanner to read from a new input stream.
   * Does not close the old reader.
   *
   * All internal variables are reset, the old input stream
   * <b>cannot</b> be reused (internal buffer is discarded and lost).
   * Lexical state is set to <tt>ZZ_INITIAL</tt>.
   *
   * @param reader   the new input stream
   */
  public final void yyreset(java.io.Reader reader) {
    zzReader = reader;
    zzAtBOL  = true;
    zzAtEOF  = false;
    zzEOFDone = false;
    zzEndRead = zzStartRead = 0;
    zzCurrentPos = zzMarkedPos = 0;
    yyline = yychar = yycolumn = 0;
    zzLexicalState = YYINITIAL;
  }


  /**
   * Returns the current lexical state.
   */
  public final int yystate() {
    return zzLexicalState;
  }


  /**
   * Enters a new lexical state
   *
   * @param newState the new lexical state
   */
  public final void yybegin(int newState) {
    zzLexicalState = newState;
  }


  /**
   * Returns the text matched by the current regular expression.
   */
  public final String yytext() {
    return new String( zzBuffer, zzStartRead, zzMarkedPos-zzStartRead );
  }


  /**
   * Returns the character at position <tt>pos</tt> from the
   * matched text.
   *
   * It is equivalent to yytext().charAt(pos), but faster
   *
   * @param pos the position of the character to fetch.
   *            A value from 0 to yylength()-1.
   *
   * @return the character at position pos
   */
  public final char yycharat(int pos) {
```

```
  return zzBuffer[zzStartRead+pos];
}


/**
 * Returns the length of the matched text region.
 */
public final int yylength() {
  return zzMarkedPos-zzStartRead;
}


/**
 * Reports an error that occured while scanning.
 *
 * In a wellformed scanner (no or only correct usage of
 * yypushback(int) and a match-all fallback rule) this method
 * will only be called with things that "Can't Possibly Happen".
 * If this method is called, something is seriously wrong
 * (e.g. a JFlex bug producing a faulty scanner etc.).
 *
 * Usual syntax/scanner level error handling should be done
 * in error fallback rules.
 *
 * @param   errorCode  the code of the errormessage to display
 */
private void zzScanError(int errorCode) {
  String message;
  try {
    message = ZZ_ERROR_MSG[errorCode];
  }
  catch (ArrayIndexOutOfBoundsException e) {
    message = ZZ_ERROR_MSG[ZZ_UNKNOWN_ERROR];
  }

  throw new Error(message);
}


/**
 * Pushes the specified amount of characters back into the input stream.
 *
 * They will be read again by then next call of the scanning method
 *
 * @param number  the number of characters to be read again.
 *                This number must not be greater than yylength()!
 */
public void yypushback(int number)  {
  if ( number > yylength() )
    zzScanError(ZZ_PUSHBACK_2BIG);

  zzMarkedPos -= number;
}


/**
 * Resumes scanning until the next regular expression is matched,
 * the end of input is encountered or an I/O-Error occurs.
 *
 * @return      the next token
 * @exception   java.io.IOException  if any I/O-Error occurs
 */
public Token yylex() throws java.io.IOException {
  int zzInput;
  int zzAction;

  // cached fields:
  int zzCurrentPosL;
  int zzMarkedPosL;
```



```
  int zzEndReadL = zzEndRead;
  char [] zzBufferL = zzBuffer;
  char [] zzCMapL = ZZ_CMAP;

  int [] zzTransL = ZZ_TRANS;
  int [] zzRowMapL = ZZ_ROWMAP;
  int [] zzAttrL = ZZ_ATTRIBUTE;

  while (true) {
    zzMarkedPosL = zzMarkedPos;

    boolean zzR = false;
    for (zzCurrentPosL = zzStartRead; zzCurrentPosL < zzMarkedPosL;
                                                    zzCurrentPosL++) {
      switch (zzBufferL[zzCurrentPosL]) {
      case '\u000B':
      case '\u000C':
      case '\u0085':
      case '\u2028':
      case '\u2029':
        yyline++;
        yycolumn = 0;
        zzR = false;
        break;
      case '\r':
        yyline++;
        yycolumn = 0;
        zzR = true;
        break;
      case '\n':
        if (zzR)
          zzR = false;
        else {
          yyline++;
          yycolumn = 0;
        }
        break;
      default:
        zzR = false;
        yycolumnn++;
      }
    }

    if (zzR) {
      // peek one character ahead if it is \n (if we have counted one line too
much)
      boolean zzPeek;
      if (zzMarkedPosL < zzEndReadL)
        zzPeek = zzBufferL[zzMarkedPosL] == '\n';
      else if (zzAtEOF)
        zzPeek = false;
      else {
        boolean eof = zzRefill();
        zzEndReadL = zzEndRead;
        zzMarkedPosL = zzMarkedPos;
        zzBufferL = zzBuffer;
        if (eof)
          zzPeek = false;
        else
          zzPeek = zzBufferL[zzMarkedPosL] == '\n';
      }
      if (zzPeek) yyline--;
    }
    zzAction = -1;

    zzCurrentPosL = zzCurrentPos = zzStartRead = zzMarkedPosL;

    zzState = ZZ_LEXSTATE[zzLexicalState];
```

```
    zzForAction: {
      while (true) {

        if (zzCurrentPosL < zzEndReadL)
          zzInput = zzBufferL[zzCurrentPosL++];
        else if (zzAtEOF) {
          zzInput = YYEOF;
          break zzForAction;
        }
        else {
          // store back cached positions
          zzCurrentPos  = zzCurrentPosL;
          zzMarkedPos   = zzMarkedPosL;
          boolean eof = zzRefill();
          // get translated positions and possibly new buffer
          zzCurrentPosL  = zzCurrentPos;
          zzMarkedPosL   = zzMarkedPos;
          zzBufferL      = zzBuffer;
          zzEndReadL     = zzEndRead;
          if (eof) {
            zzInput = YYEOF;
            break zzForAction;
          }
          else {
            zzInput = zzBufferL[zzCurrentPosL++];
          }
        }
        int zzNext = zzTransL[ zzRowMapL[zzState] + zzCMapL[zzInput] ];
        if (zzNext == -1) break zzForAction;
        zzState = zzNext;

        int zzAttributes = zzAttrL[zzState];
        if ( (zzAttributes & 1) == 1 ) {
          zzAction = zzState;
          zzMarkedPosL = zzCurrentPosL;
          if ( (zzAttributes & 8) == 8 ) break zzForAction;
        }

      }
    }

    // store back cached position
    zzMarkedPos = zzMarkedPosL;

    switch (zzAction < 0 ? zzAction : ZZ_ACTION[zzAction]) {
      case 17:
        { return (new Token("LBRACK", yytext(), yyline, yycolumn));
        }
      case 55: break;
      case 32:
        { return (new Token("TRUE", yytext(), yyline, yycolumn));
        }
      case 56: break;
      case 42:
        { return (new Token("LABEL", yytext(), yyline, yycolumn));
        }
      case 57: break;
      case 31:
        { return (new Token("END", yytext(), yyline, yycolumn));
        }
      case 58: break;
      case 39:
        { return (new Token("ELSE", yytext(), yyline, yycolumn));
        }
      case 59: break;
      case 18:
        { return (new Token("RBRACK", yytext(), yyline, yycolumn));
        }
```



```
      case 60: break;
      case 2:
        { return (new Token("IDENTIFIER", yytext(), yyline, yycolumn));
        }
      case 61: break;
      case 30:
        { return (new Token("AND", yytext(), yyline, yycolumn));
        }
      case 62: break;
      case 48:
        { return (new Token("REPEAT", yytext(), yyline, yycolumn));
        }
      case 63: break;
      case 34:
        { return (new Token("READ", yytext(), yyline, yycolumn));
        }
      case 64: break;
      case 51:
        { return (new Token("BOOLEAN", yytext(), yyline, yycolumn));
        }
      case 65: break;
      case 23:
        { return (new Token("OR", yytext(), yyline, yycolumn));
        }
      case 66: break;
      case 35:
        { return (new Token("REAL", yytext(), yyline, yycolumn));
        }
      case 67: break;
      case 52:
        { return (new Token("INTEGER", yytext(), yyline, yycolumn));
        }
      case 68: break;
      case 38:
        { return (new Token("GOTO", yytext(), yyline, yycolumn));
        }
      case 69: break;
      case 25:
        { return (new Token("DO", yytext(), yyline, yycolumn));
        }
      case 70: break;
      case 19:
        { return (new Token("REAL_CONSTANT", yytext(), yyline, yycolumn));
        }
      case 71: break;
      case 37:
        { return (new Token("CHAR", yytext(), yyline, yycolumn));
        }
      case 72: break;
      case 26:
        { return (new Token("IF", yytext(), yyline, yycolumn));
        }
      case 73: break;
      case 41:
        { return (new Token("UNTIL", yytext(), yyline, yycolumn));
        }
      case 74: break;
      case 22:
        { return (new Token("NE", yytext(), yyline, yycolumn));
        }
      case 75: break;
      case 36:
        { return (new Token("CASE", yytext(), yyline, yycolumn));
        }
      case 76: break;
      case 5:
        { return (new Token("EQ", yytext(), yyline, yycolumn));
        }
      case 77: break;
```

```
        case 54:
          { return (new Token("PROCEDURE", yytext(), yyline, yycolumn));
          }
        case 78: break;
        case 43:
          { return (new Token("BEGIN", yytext(), yyline, yycolumn));
          }
        case 79: break;
        case 15:
          { return (new Token("PARENT_OPEN", yytext(), yyline, yycolumn));
          }
        case 80: break;
        case 40:
          { return (new Token("ARRAY", yytext(), yyline, yycolumn));
          }
        case 81: break;
        case 8:
          { return (new Token("PLUS", yytext(), yyline, yycolumn));
          }
        case 82: break;
        case 29:
          { return (new Token("NOT", yytext(), yyline, yycolumn));
          }
        case 83: break;
        case 13:
          { return (new Token("TWO_POINTS", yytext(), yyline, yycolumn));
          }
        case 84: break;
        case 1:
          { System.out.println("Caractere Ilegal: <" + yytext() + ">");
          }
        case 85: break;
        case 33:
          { return (new Token("THEN", yytext(), yyline, yycolumn));
          }
        case 86: break;
        case 20:
          { return (new Token("LE", yytext(), yyline, yycolumn));
          }
        case 87: break;
        case 28:
          { return (new Token("CHAR_CONSTANT", yytext(), yyline, yycolumn));
          }
        case 88: break;
        case 21:
          { return (new Token("GE", yytext(), yyline, yycolumn));
          }
        case 89: break;
        case 24:
          { return (new Token("OF", yytext(), yyline, yycolumn));
          }
        case 90: break;
        case 47:
          { return (new Token("RETURN", yytext(), yyline, yycolumn));
          }
        case 91: break;
        case 27:
          { return (new Token("ATRIB", yytext(), yyline, yycolumn));
          }
        case 92: break;
        case 45:
          { return (new Token("WRITE", yytext(), yyline, yycolumn));
          }
        case 93: break;
        case 44:
          { return (new Token("FALSE", yytext(), yyline, yycolumn));
          }
        case 94: break;
        case 11:
```



```
          { return (new Token("DIV", yytext(), yyline, yycolumn));
          }
        case 95: break;
        case 53:
          { return (new Token("UMINUS", yytext(), yyline, yycolumn));
          }
        case 96: break;
        case 10:
          { return (new Token("MULT", yytext(), yyline, yycolumn));
          }
        case 97: break;
        case 6:
          { return (new Token("LT", yytext(), yyline, yycolumn));
          }
        case 98: break;
        case 16:
          { return (new Token("PARENT_CLOSE", yytext(), yyline, yycolumn));
          }
        case 99: break;
        case 7:
          { return (new Token("GT", yytext(), yyline, yycolumn));
          }
        case 100: break;
        case 14:
          { return (new Token("COMMA", yytext(), yyline, yycolumn));
          }
        case 101: break;
        case 9:
          { return (new Token("MINUS", yytext(), yyline, yycolumn));
          }
        case 102: break;
        case 46:
          { return (new Token("WHILE", yytext(), yyline, yycolumn));
          }
        case 103: break;
        case 50:
          { return (new Token("PROGRAM", yytext(), yyline, yycolumn));
          }
        case 104: break;
        case 12:
          { return (new Token("SEMI_COMMA", yytext(), yyline, yycolumn));
          }
        case 105: break;
        case 49:
          { return (new Token("DECLARE", yytext(), yyline, yycolumn));
          }
        case 106: break;
        case 3:
          { return (new Token("INTEGER_CONSTANT", yytext(), yyline, yycolumn));
          }
        case 107: break;
        case 4:
          {
          }
        case 108: break;
        default:
          if (zzInput == YYEOF && zzStartRead == zzCurrentPos) {
            zzAtEOF = true;
            return null;
          }
          else {
            zzScanError(ZZ_NO_MATCH);
          }
      }
    }
  }


}
```

```
------------------------------
--      TP3.FLEX            --
------------------------------
%%

%class Lexer
%line
%column
%type Token

%{

%}

letter = [A-Za-z]
digit = [0-9]
identifier = {letter}({letter}|{digit})*
unsigned_integer = {digit}({digit})*
sign = [+|-]?
scale_factor = "E"{sign}{unsigned_integer}
unsigned_real = {unsigned_integer}("."{digit}*)?{scale_factor}?
integer_constant = {unsigned_integer}
real_constant = {unsigned_real}
char_constant = \'[^\r\n]\'

constant = {integer_constant} | {real_constant} | {char_constant}

eq      = "="
lt      = "<"
le      = "<="
gt      = ">"
ge      = ">="
ne      = "!="
not     = "not"

plus    = "+"
minus   = "-"
or      = "or"

mult    = "*"
div     = "/"
and     = "and"
uminus  = [minus]+constant

NONNEWLINE_WHITE_SPACE_CHAR = [\ \t\b\012]
NEWLINE = \r|\n|\r\n

COMMENT = "//"[^\r\n]*{NEWLINE}

%%

<YYINITIAL> {

  "program"   { return (new Token("PROGRAM", yytext(), yyline, yycolumn)); }
  "declare"       { return (new Token("DECLARE", yytext(), yyline, yycolumn)); }
  "begin"     { return (new Token("BEGIN", yytext(), yyline, yycolumn)); }
  "do"        { return (new Token("DO", yytext(), yyline, yycolumn)); }
  "end"       { return (new Token("END", yytext(), yyline, yycolumn)); }
  "integer"   { return (new Token("INTEGER", yytext(), yyline, yycolumn)); }
  "real"      { return (new Token("REAL", yytext(), yyline, yycolumn)); }
  "boolean"   { return (new Token("BOOLEAN", yytext(), yyline, yycolumn)); }
  "char"      { return (new Token("CHAR", yytext(), yyline, yycolumn)); }
  "label"     { return (new Token("LABEL", yytext(), yyline, yycolumn)); }
  "array"     { return (new Token("ARRAY", yytext(), yyline, yycolumn)); }
  "of"        { return (new Token("OF", yytext(), yyline, yycolumn)); }
  "procedure" { return (new Token("PROCEDURE", yytext(), yyline, yycolumn)); }
```



```
  "if"        { return (new Token("IF", yytext(), yyline, yycolumn)); }
  "then"      { return (new Token("THEN", yytext(), yyline, yycolumn)); }
  "else"      { return (new Token("ELSE", yytext(), yyline, yycolumn)); }
  "case"      { return (new Token("CASE", yytext(), yyline, yycolumn)); }
  "repeat"    { return (new Token("REPEAT", yytext(), yyline, yycolumn)); }
  "while"     { return (new Token("WHILE", yytext(), yyline, yycolumn)); }
  "until"     { return (new Token("UNTIL", yytext(), yyline, yycolumn)); }
  "read"      { return (new Token("READ", yytext(), yyline, yycolumn)); }
  "write"     { return (new Token("WRITE", yytext(), yyline, yycolumn)); }
  "goto"      { return (new Token("GOTO", yytext(), yyline, yycolumn)); }
  "return"    { return (new Token("RETURN", yytext(), yyline, yycolumn)); }
  "false"     { return (new Token("FALSE", yytext(), yyline, yycolumn)); }
  "true"      { return (new Token("TRUE", yytext(), yyline, yycolumn)); }

  ";"         { return (new Token("SEMI_COMMA", yytext(), yyline, yycolumn)); }
  ":"         { return (new Token("TWO_POINTS", yytext(), yyline, yycolumn)); }
  ","         { return (new Token("COMMA", yytext(), yyline, yycolumn)); }
  ":="        { return (new Token("ATRIB", yytext(), yyline, yycolumn)); }
  "("         { return (new Token("PARENT_OPEN", yytext(), yyline, yycolumn)); }
  ")"         { return (new Token("PARENT_CLOSE", yytext(), yyline, yycolumn));
}
  "["         { return (new Token("LBRACK", yytext(), yyline, yycolumn)); }
  "]"         { return (new Token("RBRACK", yytext(), yyline, yycolumn)); }


  {COMMENT}   { }

  /* operadores de relacao */
  {eq}        { return (new Token("EQ", yytext(), yyline, yycolumn)); }
  {lt}        { return (new Token("LT", yytext(), yyline, yycolumn)); }
  {le}        { return (new Token("LE", yytext(), yyline, yycolumn)); }
  {gt}        { return (new Token("GT", yytext(), yyline, yycolumn)); }
  {ge}        { return (new Token("GE", yytext(), yyline, yycolumn)); }
  {ne}        { return (new Token("NE", yytext(), yyline, yycolumn)); }
  {not}       { return (new Token("NOT", yytext(), yyline, yycolumn)); }

  /* operadores de adicao */
  {plus}      { return (new Token("PLUS", yytext(), yyline, yycolumn)); }
  {minus}     { return (new Token("MINUS", yytext(), yyline, yycolumn)); }
  {or}        { return (new Token("OR", yytext(), yyline, yycolumn)); }

  /* operadores de multiplicacao */
  {mult}      { return (new Token("MULT", yytext(), yyline, yycolumn)); }
  {div}       { return (new Token("DIV", yytext(), yyline, yycolumn)); }
  {and}       { return (new Token("AND", yytext(), yyline, yycolumn)); }
  {uminus}    { return (new Token("UMINUS", yytext(), yyline, yycolumn)); }

  {identifier}       { return (new Token("IDENTIFIER", yytext(), yyline, yycolum
n)); }
  {integer_constant} { return (new Token("INTEGER_CONSTANT", yytext(), yyline, y
ycolumn)); }
  {real_constant}    { return (new Token("REAL_CONSTANT", yytext(), yyline, yyco
lumn)); }
  {char_constant}    { return (new Token("CHAR_CONSTANT", yytext(), yyline, yyco
lumn)); }

  {NONNEWLINE_WHITE_SPACE_CHAR}+ { }

}

{NEWLINE} { }

. {
  System.out.println("Caractere Ilegal: <" + yytext() + ">");
}
```