```
// DO NOT EDIT
// Generated by JFlex 1.9.1 http://jflex.de/
// source: src/analizadores/Scanner.jflex

package analizadores;
import java_cup.runtime.Symbol;
import excepciones.Errores;
import analizadores.TokenLista; // IMPORTANTE: Para guardar los tokens
import analizadores.Tokens;     // IMPORTANTE: Tu enum de tokens


@SuppressWarnings("fallthrough")
public class Scanner implements java_cup.runtime.Scanner {

 /** 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
 };

 /**
  * Top-level table for translating characters to character classes
  */
 private static final int [] ZZ_CMAP_TOP = zzUnpackcmap_top();

 private static final String ZZ_CMAP_TOP_PACKED_0 =
   "\1\0\1\u0100\36\u0200\1\u0300\1\u0400\266\u0200\10\u0500\u1020\u0200";

 private static int [] zzUnpackcmap_top() {
   int [] result = new int[4352];
   int offset = 0;
   offset = zzUnpackcmap_top(ZZ_CMAP_TOP_PACKED_0, offset, result);
   return result;
 }

 private static int zzUnpackcmap_top(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;
}


/**
 * Second-level tables for translating characters to character classes
 */
private static final int [] ZZ_CMAP_BLOCKS = zzUnpackcmap_blocks();

private static final String ZZ_CMAP_BLOCKS_PACKED_0 =
  "\11\0\1\1\1\2\1\3\2\2\22\0\1\1\1\4"+
  "\1\5\2\0\1\6\1\7\1\10\1\11\1\12\1\13"+
  "\1\14\1\15\1\16\1\17\1\20\12\21\1\22\1\23"+
  "\1\24\1\25\1\26\2\0\1\27\1\30\1\31\1\32"+
  "\1\33\1\34\1\35\1\36\1\37\1\40\1\41\1\42"+
  "\1\40\1\43\1\44\1\45\1\40\1\46\1\47\1\50"+
  "\1\51\1\52\1\53\3\40\1\54\1\55\1\56\1\57"+
  "\1\60\1\0\1\27\1\30\1\31\1\32\1\33\1\34"+
  "\1\35\1\36\1\37\1\40\1\41\1\42\1\40\1\43"+
  "\1\44\1\45\1\40\1\46\1\47\1\50\1\51\1\52"+
  "\1\53\3\40\1\61\1\62\1\63\7\0\1\3\252\0"+
  "\2\64\115\0\1\65\u01a8\0\2\3\u0100\0\1\66\325\0"+
  "\u0100\3";

private static int [] zzUnpackcmap_blocks() {
  int [] result = new int[1536];
  int offset = 0;
  offset = zzUnpackcmap_blocks(ZZ_CMAP_BLOCKS_PACKED_0, offset, result);
  return result;
}

private static int zzUnpackcmap_blocks(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 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\2\1\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\20\23\1\24\1\25"+
  "\1\26\1\27\1\1\1\30\2\1\1\31\1\0\1\32"+
  "\1\0\1\33\2\0\1\34\1\35\1\36\1\0\1\2"+
  "\1\0\1\37\1\40\1\41\7\23\1\42\3\23\1\43"+
  "\2\23\1\0\10\23\1\44\1\43\3\0\1\45\1\0"+
  "\1\46\4\23\1\0\5\23\1\0\1\23\1\47\1\50"+
```

```
  "\1\23\1\0\1\51\1\23\1\0\4\23\1\0\1\23"+
  "\1\52\1\23\1\0\1\50\2\0\1\2\1\23\1\53"+
  "\1\23\2\54\1\55\3\23\2\56\1\23\1\0\2\57"+
  "\1\23\1\0\3\23\1\0\1\23\1\0\1\60\1\23"+
  "\2\0\1\23\2\61\1\23\1\0\2\23\2\62\2\63"+
  "\1\23\1\64\1\23\1\0\1\23\1\0\2\65\1\64"+
  "\1\66\1\23\1\0\1\23\1\67\1\70\2\71\2\72"+
  "\1\23\1\0\1\73\2\74";

private static int [] zzUnpackAction() {
  int [] result = new int[187];
  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\67\0\156\0\245\0\334\0\67\0\u0113\0\u014a"+
  "\0\67\0\67\0\u0181\0\u01b8\0\67\0\u01ef\0\67\0\u0226"+
  "\0\u025d\0\67\0\67\0\u0294\0\u02cb\0\u0302\0\u0339\0\u0370"+
  "\0\u03a7\0\u03de\0\u0415\0\u044c\0\u0483\0\u04ba\0\u04f1\0\u0528"+
  "\0\u055f\0\u0596\0\u05cd\0\u0604\0\u063b\0\u0672\0\67\0\67"+
  "\0\67\0\67\0\u06a9\0\67\0\u06e0\0\u0717\0\67\0\334"+
  "\0\67\0\u074e\0\67\0\u0785\0\u07bc\0\67\0\67\0\67"+
  "\0\u07f3\0\u082a\0\u0861\0\67\0\67\0\67\0\u0898\0\u08cf"+
  "\0\u0906\0\u093d\0\u0974\0\u09ab\0\u09e2\0\u0a19\0\u0a50\0\u0a87"+
  "\0\u0abe\0\u0483\0\u0af5\0\u0b2c\0\u0b63\0\u0b9a\0\u0bd1\0\u0c08"+
  "\0\u0c3f\0\u0c76\0\u0cad\0\u0ce4\0\u0d1b\0\67\0\67\0\u0d52"+
  "\0\u0d89\0\u0dc0\0\67\0\u0df7\0\u0861\0\u0e2e\0\u0e65\0\u0e9c"+
  "\0\u0ed3\0\u0f0a\0\u0f41\0\u0f78\0\u0faf\0\u0fe6\0\u101d\0\u1054"+
  "\0\u108b\0\u0483\0\u0483\0\u10c2\0\u10f9\0\u0483\0\u1130\0\u1167"+
  "\0\u119e\0\u11d5\0\u120c\0\u1243\0\u127a\0\u12b1\0\u0483\0\u12e8"+
  "\0\u131f\0\67\0\u1356\0\u138d\0\u07f3\0\u13c4\0\u0483\0\u13fb"+
  "\0\u0483\0\67\0\u0483\0\u1432\0\u1469\0\u14a0\0\u0483\0\67"+
  "\0\u14d7\0\u150e\0\u0483\0\67\0\u1545\0\u157c\0\u15b3\0\u15ea"+
  "\0\u1621\0\u1658\0\u168f\0\u16c6\0\u0483\0\u16fd\0\u1734\0\u176b"+
  "\0\u17a2\0\u0483\0\67\0\u17d9\0\u1810\0\u1847\0\u187e\0\u0483"+
  "\0\67\0\u0483\0\67\0\u18b5\0\u0483\0\u18ec\0\u1923\0\u195a"+
  "\0\u1991\0\u0483\0\67\0\67\0\u0483\0\u19c8\0\u19ff\0\u1a36"+
```

```
  "\0\u0483\0\u0483\0\u0483\0\67\0\u0483\0\67\0\u1a6d\0\u1aa4"+
  "\0\u0483\0\u0483\0\67";

private static int [] zzUnpackRowMap() {
  int [] result = new int[187];
  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() - 1;
  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\2\3\1\2\1\4\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\1\30"+
  "\1\31\1\32\1\33\1\34\2\35\1\36\2\35\1\37"+
  "\1\40\1\35\1\41\1\42\1\43\1\44\1\35\1\45"+
  "\1\46\1\47\1\2\1\50\1\51\1\2\1\52\1\53"+
  "\1\54\1\55\1\56\1\2\70\0\2\3\111\0\1\57"+
  "\41\0\5\60\1\61\47\60\1\62\11\60\7\0\1\63"+
  "\57\0\10\64\1\0\44\64\1\65\11\64\13\0\1\66"+
  "\67\0\1\67\70\0\1\70\63\0\1\71\4\0\1\72"+
  "\65\0\1\73\1\0\1\21\72\0\1\74\66\0\1\75"+
  "\66\0\1\76\62\0\1\35\5\0\16\35\1\77\6\35"+
  "\4\0\1\35\27\0\1\35\5\0\15\35\1\100\1\35"+
  "\1\101\5\35\4\0\1\35\27\0\1\35\5\0\1\102"+
  "\6\35\1\103\5\35\1\104\7\35\4\0\1\35\27\0"+
  "\1\35\5\0\4\35\1\105\10\35\1\106\7\35\4\0"+
  "\1\35\27\0\1\35\5\0\13\35\1\107\11\35\4\0"+
  "\1\35\27\0\1\35\5\0\1\110\14\35\1\111\7\35"+
  "\4\0\1\35\27\0\1\35\5\0\25\35\4\0\1\35"+
  "\27\0\1\35\5\0\5\35\1\112\6\35\1\113\10\35"+
  "\4\0\1\35\27\0\1\35\5\0\10\35\1\114\14\35"+
  "\4\0\1\35\3\0\1\115\23\0\1\35\5\0\4\35"+
  "\1\116\20\35\4\0\1\35\27\0\1\35\5\0\17\35"+
  "\1\117\5\35\4\0\1\35\27\0\1\35\5\0\4\35"+
  "\1\120\20\35\4\0\1\35\27\0\1\35\5\0\21\35"+
  "\1\121\2\35\1\122\4\0\1\35\27\0\1\35\5\0"+
  "\17\35\1\123\5\35\4\0\1\35\27\0\1\35\5\0"+
  "\1\124\24\35\4\0\1\35\27\0\1\35\5\0\7\35"+
  "\1\125\15\35\4\0\1\35\70\0\1\126\40\0\1\127"+
  "\6\0\1\130\73\0\1\131\2\0\1\132\13\0\2\60"+
```

```
"\2\0\63\60\10\0\1\133\56\0\2\64\2\0\63\64"+
"\13\71\1\134\53\71\2\72\2\0\63\72\21\0\1\135"+
"\66\0\1\35\5\0\16\35\1\136\6\35\4\0\1\35"+
"\27\0\1\35\5\0\15\35\1\137\7\35\4\0\1\35"+
"\27\0\1\35\5\0\4\35\1\140\20\35\4\0\1\35"+
"\27\0\1\35\5\0\20\35\1\141\4\35\4\0\1\35"+
"\4\0\1\142\22\0\1\35\5\0\1\143\24\35\4\0"+
"\1\35\27\0\1\35\5\0\14\35\1\144\10\35\4\0"+
"\1\35\27\0\1\35\5\0\5\35\1\145\17\35\4\0"+
"\1\35\27\0\1\35\5\0\22\35\1\146\2\35\4\0"+
"\1\35\27\0\1\35\5\0\20\35\1\147\4\35\4\0"+
"\1\35\4\0\1\150\22\0\1\35\5\0\13\35\1\151"+
"\11\35\4\0\1\35\27\0\1\35\5\0\17\35\1\152"+
"\5\35\4\0\1\35\27\0\1\35\5\0\21\35\1\153"+
"\3\35\4\0\1\35\27\0\1\35\5\0\20\35\1\154"+
"\4\35\4\0\1\35\4\0\1\155\50\0\1\155\15\0"+
"\1\155\22\0\1\35\5\0\24\35\1\156\4\0\1\35"+
"\27\0\1\35\5\0\10\35\1\157\14\35\4\0\1\35"+
"\3\0\1\160\23\0\1\35\5\0\21\35\1\161\3\35"+
"\4\0\1\35\27\0\1\35\5\0\1\162\16\35\1\163"+
"\5\35\4\0\1\35\27\0\1\35\5\0\10\35\1\164"+
"\14\35\4\0\1\35\3\0\1\165\23\0\1\35\5\0"+
"\22\35\1\166\2\35\4\0\1\35\27\0\1\35\5\0"+
"\17\35\1\167\5\35\4\0\1\35\27\0\1\35\5\0"+
"\10\35\1\170\14\35\4\0\1\35\3\0\1\171\52\0"+
"\1\172\45\0\1\173\16\0\1\174\57\0\1\165\24\0"+
"\1\165\2\0\13\71\1\134\4\71\1\175\46\71\21\0"+
"\1\35\5\0\4\35\1\176\20\35\4\0\1\35\27\0"+
"\1\35\5\0\13\35\1\177\11\35\4\0\1\35\27\0"+
"\1\35\5\0\1\200\24\35\4\0\1\35\27\0\1\35"+
"\5\0\4\35\1\201\20\35\4\0\1\35\41\0\1\202"+
"\54\0\1\35\5\0\17\35\1\203\5\35\4\0\1\35"+
"\27\0\1\35\5\0\21\35\1\204\3\35\4\0\1\35"+
"\27\0\1\35\5\0\1\205\24\35\4\0\1\35\27\0"+
"\1\35\5\0\1\35\1\206\23\35\4\0\1\35\27\0"+
"\1\35\5\0\4\35\1\207\20\35\4\0\1\35\41\0"+
"\1\210\54\0\1\35\5\0\20\35\1\211\4\35\4\0"+
"\1\35\4\0\1\212\22\0\1\35\5\0\21\35\1\213"+
"\3\35\4\0\1\35\56\0\1\214\37\0\1\35\5\0"+
"\14\35\1\215\10\35\4\0\1\35\51\0\1\216\44\0"+
"\1\35\5\0\22\35\1\217\2\35\4\0\1\35\27\0"+
"\1\35\5\0\17\35\1\220\5\35\4\0\1\35\27\0"+
"\1\35\5\0\10\35\1\221\14\35\4\0\1\35\3\0"+
"\1\222\23\0\1\35\5\0\21\35\1\223\3\35\4\0"+
"\1\35\56\0\1\224\37\0\1\35\5\0\4\35\1\225"+
"\20\35\4\0\1\35\27\0\1\35\5\0\13\35\1\226"+
"\11\35\4\0\1\35\50\0\1\227\72\0\1\230\57\0"+
"\1\222\24\0\1\222\23\0\1\35\5\0\14\35\1\231"+
"\10\35\4\0\1\35\27\0\1\35\5\0\12\35\1\232"+
"\12\35\4\0\1\35\5\0\1\233\21\0\1\35\5\0"+
"\10\35\1\234\14\35\4\0\1\35\3\0\1\235\23\0"+
"\1\35\5\0\22\35\1\236\2\35\4\0\1\35\27\0"+
"\1\35\5\0\13\35\1\237\11\35\4\0\1\35\27\0"+
"\1\35\5\0\4\35\1\240\20\35\4\0\1\35\41\0"+
"\1\241\54\0\1\35\5\0\21\35\1\242\3\35\4\0"+
"\1\35\56\0\1\243\37\0\1\35\5\0\17\35\1\244"+
"\5\35\4\0\1\35\27\0\1\35\5\0\21\35\1\245"+
```

```
  "\3\35\4\0\1\35\27\0\1\35\5\0\14\35\1\246"+
  "\10\35\4\0\1\35\51\0\1\247\44\0\1\35\5\0"+
  "\2\35\1\250\22\35\4\0\1\35\37\0\1\251\56\0"+
  "\1\35\5\0\4\35\1\252\20\35\4\0\1\35\41\0"+
  "\1\253\103\0\1\254\37\0\1\35\5\0\3\35\1\255"+
  "\21\35\4\0\1\35\27\0\1\35\5\0\14\35\1\256"+
  "\10\35\4\0\1\35\51\0\1\257\44\0\1\35\5\0"+
  "\13\35\1\260\11\35\4\0\1\35\27\0\1\35\5\0"+
  "\4\35\1\261\20\35\4\0\1\35\27\0\1\35\5\0"+
  "\14\35\1\262\10\35\4\0\1\35\27\0\1\35\5\0"+
  "\6\35\1\263\16\35\4\0\1\35\43\0\1\264\52\0"+
  "\1\35\5\0\7\35\1\265\15\35\4\0\1\35\44\0"+
  "\1\266\51\0\1\35\5\0\22\35\1\267\2\35\4\0"+
  "\1\35\57\0\1\270\36\0\1\35\5\0\21\35\1\271"+
  "\3\35\4\0\1\35\27\0\1\35\5\0\4\35\1\272"+
  "\20\35\4\0\1\35\41\0\1\273\33\0";

private static int [] zzUnpacktrans() {
  int [] result = new int[6875];
  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 code for "Unknown internal scanner error". */
private static final int ZZ_UNKNOWN_ERROR = 0;
/** Error code for "could not match input". */
private static final int ZZ_NO_MATCH = 1;
/** Error code for "pushback value was too large". */
private static final int ZZ_PUSHBACK_2BIG = 2;

/**
 * Error messages for {@link #ZZ_UNKNOWN_ERROR}, {@link #ZZ_NO_MATCH}, and
 * {@link #ZZ_PUSHBACK_2BIG} respectively.
 */
private static final String ZZ_ERROR_MSG[] = {
  "Unknown internal scanner error",
  "Error: could not match input",
  "Error: pushback value was too large"
};

/**
 * ZZ_ATTRIBUTE[aState] contains the attributes of state {@code aState}
```

```
 */
private static final int [] ZZ_ATTRIBUTE = zzUnpackAttribute();

private static final String ZZ_ATTRIBUTE_PACKED_0 =
  "\1\0\1\11\3\1\1\11\2\1\2\11\2\1\1\11"+
  "\1\1\1\11\2\1\2\11\23\1\4\11\1\1\1\11"+
  "\2\1\1\11\1\0\1\11\1\0\1\11\2\0\3\11"+
  "\1\0\1\1\1\0\3\11\16\1\1\0\10\1\2\11"+
  "\3\0\1\11\1\0\5\1\1\0\5\1\1\0\4\1"+
  "\1\0\2\1\1\0\4\1\1\0\3\1\1\0\1\11"+
  "\2\0\5\1\1\11\5\1\1\11\1\1\1\0\1\1"+
  "\1\11\1\1\1\0\3\1\1\0\1\1\1\0\2\1"+
  "\2\0\2\1\1\11\1\1\1\0\3\1\1\11\1\1"+
  "\1\11\3\1\1\0\1\1\1\0\1\1\2\11\2\1"+
  "\1\0\4\1\1\11\1\1\1\11\1\1\1\0\2\1"+
  "\1\11";

private static int [] zzUnpackAttribute() {
  int [] result = new int[187];
  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;
}

/** Input device. */
private java.io.Reader zzReader;

/** Current state of the DFA. */
private int zzState;

/** Current lexical state. */
private int zzLexicalState = YYINITIAL;

/**
 * This buffer contains the current text to be matched and is the source of the {@link #yytext()}
 * string.
 */
private char zzBuffer[] = new char[Math.min(ZZ_BUFFERSIZE, zzMaxBufferLen())];

/** Text position at the last accepting state. */
private int zzMarkedPos;

/** Current text position in the buffer. */
private int zzCurrentPos;
```

```
/** Marks the beginning of the {@link #yytext()} string in the buffer. */
private int zzStartRead;

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

/**
 * Whether the scanner is at the end of file.
 * @see #yyatEOF
 */
private boolean zzAtEOF;

/**
 * The number of occupied positions in {@link #zzBuffer} beyond {@link #zzEndRead}.
 *
 * <p>When a lead/high surrogate has been read from the input stream into the final
 * {@link #zzBuffer} position, this will have a value of 1; otherwise, it will have a value of 0.
 */
private int zzFinalHighSurrogate = 0;

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

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

/** Number of characters up to the start of the matched text. */
@SuppressWarnings("unused")
private long yychar;

/** Whether the scanner is currently at the beginning of a line. */
@SuppressWarnings("unused")
private boolean zzAtBOL = true;

/** Whether the user-EOF-code has already been executed. */
private boolean zzEOFDone;

/* user code: */
  // --- MÉTODOS AUXILIARES PARA GUARDAR EN LA LISTA ---

  // 1. Para tokens con valor (como números o IDs)
  private Symbol symbol(int type, Object value, Tokens tokenTipo) {
     TokenLista.guardar(tokenTipo, yytext(), yyline+1, yycolumn+1);
     return new Symbol(type, yyline+1, yycolumn+1, value);
  }

  // 2. Para tokens simples (palabras reservadas, signos)
  private Symbol symbol(int type, Tokens tokenTipo) {
     TokenLista.guardar(tokenTipo, yytext(), yyline+1, yycolumn+1);
     return new Symbol(type, yyline+1, yycolumn+1);
  }


/**
 * Creates a new scanner
 *
 * @param   in  the java.io.Reader to read input from.
```

```
 */
public Scanner(java.io.Reader in) {
  this.zzReader = in;
}


/** Returns the maximum size of the scanner buffer, which limits the size of tokens. */
private int zzMaxBufferLen() {
  return Integer.MAX_VALUE;
}

/**  Whether the scanner buffer can grow to accommodate a larger token. */
private boolean zzCanGrow() {
  return true;
}

/**
 * Translates raw input code points to DFA table row
 */
private static int zzCMap(int input) {
  int offset = input & 255;
  return offset == input ? ZZ_CMAP_BLOCKS[offset] : ZZ_CMAP_BLOCKS[ZZ_CMAP_TOP[input >> 8] |
offset];
}

/**
 * Refills the input buffer.
 *
 * @return {@code false} 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) {
    zzEndRead += zzFinalHighSurrogate;
    zzFinalHighSurrogate = 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 - zzFinalHighSurrogate && zzCanGrow()) {
    /* if not, and it can grow: blow it up */
    char newBuffer[] = new char[Math.min(zzBuffer.length * 2, zzMaxBufferLen())];
    System.arraycopy(zzBuffer, 0, newBuffer, 0, zzBuffer.length);
    zzBuffer = newBuffer;
    zzEndRead += zzFinalHighSurrogate;
    zzFinalHighSurrogate = 0;
  }
```

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

  /* not supposed to occur according to specification of java.io.Reader */
  if (numRead == 0) {
    if (requested == 0) {
      throw new java.io.EOFException("Scan buffer limit reached ["+zzBuffer.length+"]");
    }
    else {
      throw new java.io.IOException(
          "Reader returned 0 characters. See JFlex examples/zero-reader for a workaround.");
    }
  }
  if (numRead > 0) {
    zzEndRead += numRead;
    if (Character.isHighSurrogate(zzBuffer[zzEndRead - 1])) {
      if (numRead == requested) { // We requested too few chars to encode a full Unicode character
        --zzEndRead;
        zzFinalHighSurrogate = 1;
      } else {                    // There is room in the buffer for at least one more char
        int c = zzReader.read();  // Expecting to read a paired low surrogate char
        if (c == -1) {
          return true;
        } else {
          zzBuffer[zzEndRead++] = (char)c;
        }
      }
    }
    /* potentially more input available */
    return false;
  }

  /* numRead < 0 ==> end of stream */
  return true;
}


/**
 * Closes the input reader.
 *
 * @throws java.io.IOException if the reader could not be closed.
 */
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.
 *
```

```
 * <p>Does not close the old reader.
 *
 * <p>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 {@code ZZ_INITIAL}.
 *
 * <p>Internal scan buffer is resized down to its initial length, if it has grown.
 *
 * @param reader The new input stream.
 */
public final void yyreset(java.io.Reader reader) {
  zzReader = reader;
  zzEOFDone = false;
  yyResetPosition();
  zzLexicalState = YYINITIAL;
  int initBufferSize = Math.min(ZZ_BUFFERSIZE, zzMaxBufferLen());
  if (zzBuffer.length > initBufferSize) {
    zzBuffer = new char[initBufferSize];
  }
}

/**
 * Resets the input position.
 */
private final void yyResetPosition() {
    zzAtBOL  = true;
    zzAtEOF  = false;
    zzCurrentPos = 0;
    zzMarkedPos = 0;
    zzStartRead = 0;
    zzEndRead = 0;
    zzFinalHighSurrogate = 0;
    yyline = 0;
    yycolumn = 0;
    yychar = 0L;
}


/**
 * Returns whether the scanner has reached the end of the reader it reads from.
 *
 * @return whether the scanner has reached EOF.
 */
public final boolean yyatEOF() {
  return zzAtEOF;
}


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


/**
 * Returns the character at the given position from the matched text.
 *
 * <p>It is equivalent to {@code yytext().charAt(pos)}, but faster.
 *
 * @param position the position of the character to fetch. A value from 0 to {@code yylength()-1}.
 *
 * @return the character at {@code position}.
 */
public final char yycharat(int position) {
  return zzBuffer[zzStartRead + position];
}


/**
 * How many characters were matched.
 *
 * @return the length of the matched text region.
 */
public final int yylength() {
  return zzMarkedPos-zzStartRead;
}


/**
 * Reports an error that occurred while scanning.
 *
 * <p>In a well-formed scanner (no or only correct usage of {@code yypushback(int)} and a
 * match-all fallback rule) this method will only be called with things that
 * "Can't Possibly Happen".
 *
 * <p>If this method is called, something is seriously wrong (e.g. a JFlex bug producing a faulty
 * scanner etc.).
 *
 * <p>Usual syntax/scanner level error handling should be done in error fallback rules.
 *
 * @param errorCode the code of the error message to display.
```

```
 */
 private static 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.
  *
  * <p>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
  *     {@link #yylength()}.
  */
 public void yypushback(int number)  {
  if ( number > yylength() )
    zzScanError(ZZ_PUSHBACK_2BIG);

  zzMarkedPos -= number;
 }


 /**
  * Contains user EOF-code, which will be executed exactly once,
  * when the end of file is reached
  */
 private void zzDoEOF() throws java.io.IOException {
  if (!zzEOFDone) {
    zzEOFDone = true;

 yyclose();    }
 }




 /**
  * 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.
  */
 @Override  public java_cup.runtime.Symbol next_token() throws java.io.IOException
 {
  int zzInput;
  int zzAction;

  // cached fields:
```

```
int zzCurrentPosL;
int zzMarkedPosL;
int zzEndReadL = zzEndRead;
char[] zzBufferL = zzBuffer;

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

while (true) {
  zzMarkedPosL = zzMarkedPos;

  boolean zzR = false;
  int zzCh;
  int zzCharCount;
  for (zzCurrentPosL = zzStartRead  ;
       zzCurrentPosL < zzMarkedPosL ;
       zzCurrentPosL += zzCharCount ) {
    zzCh = Character.codePointAt(zzBufferL, zzCurrentPosL, zzMarkedPosL);
    zzCharCount = Character.charCount(zzCh);
    switch (zzCh) {
    case '\u000B':  // fall through
    case '\u000C':  // fall through
    case '\u0085':  // fall through
    case '\u2028':  // fall through
    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;
      yycolumn += zzCharCount;
    }
  }

  if (zzR) {
    // peek one character ahead if it is
    // (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];

// set up zzAction for empty match case:
int zzAttributes = zzAttrL[zzState];
if ( (zzAttributes & 1) == 1 ) {
  zzAction = zzState;
}


zzForAction: {
  while (true) {

    if (zzCurrentPosL < zzEndReadL) {
      zzInput = Character.codePointAt(zzBufferL, zzCurrentPosL, zzEndReadL);
      zzCurrentPosL += Character.charCount(zzInput);
    }
    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 = Character.codePointAt(zzBufferL, zzCurrentPosL, zzEndReadL);
        zzCurrentPosL += Character.charCount(zzInput);
      }
    }
    int zzNext = zzTransL[ zzRowMapL[zzState] + zzCMap(zzInput) ];
    if (zzNext == -1) break zzForAction;
```

```
    zzState = zzNext;

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

  }
 }

 // store back cached position
 zzMarkedPos = zzMarkedPosL;

 if (zzInput == YYEOF && zzStartRead == zzCurrentPos) {
  zzAtEOF = true;
      zzDoEOF();
    { return new java_cup.runtime.Symbol(sym.EOF); }
 }
 else {
  switch (zzAction < 0 ? zzAction : ZZ_ACTION[zzAction]) {
    case 1:
      { Errores.agregar("Lexico", "Caracter no reconocido: " + yytext(), yyline+1, yycolumn+1);
TokenLista.guardar(Tokens.ERROR, yytext(), yyline+1, yycolumn+1); // Guardamos el error también
System.err.println("Error léxico: " + yytext() + " en línea " + (yyline+1));
      }
    // fall through
    case 61: break;
    case 2:
      { /* Ignorar */
      }
    // fall through
    case 62: break;
    case 3:
      { return symbol(sym.NOT, Tokens.NOT);
      }
    // fall through
    case 63: break;
    case 4:
      { return symbol(sym.MOD, Tokens.MOD);
      }
    // fall through
    case 64: break;
    case 5:
      { return symbol(sym.PAR_A, Tokens.PAR_A);
      }
    // fall through
    case 65: break;
    case 6:
      { return symbol(sym.PAR_C, Tokens.PAR_C);
      }
    // fall through
    case 66: break;
    case 7:
      { return symbol(sym.POR, Tokens.MULT);
      }
```

```
// fall through
case 67: break;
case 8:
  { return symbol(sym.MAS, Tokens.SUMA);
  }
// fall through
case 68: break;
case 9:
  { return symbol(sym.COMA, Tokens.COMA);
  }
// fall through
case 69: break;
case 10:
  { return symbol(sym.MENOS, Tokens.RESTA);
  }
// fall through
case 70: break;
case 11:
  { return symbol(sym.PUNTO, Tokens.PUNTO);
  }
// fall through
case 71: break;
case 12:
  { return symbol(sym.DIV, Tokens.DIV);
  }
// fall through
case 72: break;
case 13:
  { return symbol(sym.ENTERO, yytext(), Tokens.ENTERO);
  }
// fall through
case 73: break;
case 14:
  { return symbol(sym.DOSPTOS, Tokens.DOS_PUNTOS);
  }
// fall through
case 74: break;
case 15:
  { return symbol(sym.PTCOMA, Tokens.PUNTO_COMA);
  }
// fall through
case 75: break;
case 16:
  { return symbol(sym.MENOR, Tokens.MENOR);
  }
// fall through
case 76: break;
case 17:
  { return symbol(sym.IGUAL, Tokens.ASIGN);
  }
// fall through
case 77: break;
case 18:
  { return symbol(sym.MAYOR, Tokens.MAYOR);
  }
// fall through
case 78: break;
```

```
 case 19:
  { return symbol(sym.ID, yytext(), Tokens.ID);
  }
 // fall through
 case 79: break;
 case 20:
  { return symbol(sym.COR_IZQ, Tokens.COR_IZQ);
  }
 // fall through
 case 80: break;
 case 21:
  { return symbol(sym.COR_DER, Tokens.COR_DER);
  }
 // fall through
 case 81: break;
 case 22:
  { return symbol(sym.XOR, Tokens.XOR);
  }
 // fall through
 case 82: break;
 case 23:
  { return symbol(sym.LLAVE_A, Tokens.LLAVE_A);
  }
 // fall through
 case 83: break;
 case 24:
  { return symbol(sym.LLAVE_C, Tokens.LLAVE_C);
  }
 // fall through
 case 84: break;
 case 25:
  { return symbol(sym.DIFERENTE, Tokens.DIF);
  }
 // fall through
 case 85: break;
 case 26:
  { String val = yytext();
// Guardamos las comillas y todo en el reporte
TokenLista.guardar(Tokens.LIT_STRING, val, yyline+1, yycolumn+1);
// Al parser le mandamos el valor limpio (sin comillas)
return new Symbol(sym.LIT_STRING, yyline+1, yycolumn+1, val.substring(1, val.length()-1));
  }
 // fall through
 case 86: break;
 case 27:
  { return symbol(sym.AND, Tokens.AND);
  }
 // fall through
 case 87: break;
 case 28:
  { return symbol(sym.POTENCIA, Tokens.POT);
  }
 // fall through
 case 88: break;
 case 29:
  { return symbol(sym.MASMAS, Tokens.INCREMENTO);
  }
```

```
 // fall through
 case 89: break;
 case 30:
   { return symbol(sym.MENOSMENOS, Tokens.DECREMENTO);
   }
 // fall through
 case 90: break;
 case 31:
   { return symbol(sym.MENORIGUAL, Tokens.MENORIGUAL);
   }
 // fall through
 case 91: break;
 case 32:
   { return symbol(sym.IGUALIGUAL, Tokens.IGUAL);
   }
 // fall through
 case 92: break;
 case 33:
   { return symbol(sym.MAYORIGUAL, Tokens.MAYORIGUAL);
   }
 // fall through
 case 93: break;
 case 34:
   { return symbol(sym.DO, Tokens.DO);
   }
 // fall through
 case 94: break;
 case 35:
   { return symbol(sym.IF, Tokens.IF);
   }
 // fall through
 case 95: break;
 case 36:
   { return symbol(sym.OR, Tokens.OR);
   }
 // fall through
 case 96: break;
 case 37:
   { String val = yytext();
TokenLista.guardar(Tokens.LIT_CHAR, val, yyline+1, yycolumn+1);
return new Symbol(sym.LIT_CHAR, yyline+1, yycolumn+1, val.substring(1, val.length()-1));
   }
 // fall through
 case 97: break;
 case 38:
   { return symbol(sym.DECIMAL, yytext(), Tokens.DECIMAL);
   }
 // fall through
 case 98: break;
 case 39:
   { return symbol(sym.FOR, Tokens.FOR);
   }
 // fall through
 case 99: break;
 case 40:
   { return symbol(sym.INT, Tokens.INT);
   }
```

```
// fall through
case 100: break;
case 41:
  { return symbol(sym.NEW, Tokens.NEW);
  }
// fall through
case 101: break;
case 42:
  { return symbol(sym.VAR, Tokens.VAR);
  }
// fall through
case 102: break;
case 43:
  { return symbol(sym.BOOL, Tokens.BOOL);
  }
// fall through
case 103: break;
case 44:
  { return symbol(sym.CASE, Tokens.CASE);
  }
// fall through
case 104: break;
case 45:
  { return symbol(sym.CHAR, Tokens.CHAR);
  }
// fall through
case 105: break;
case 46:
  { return symbol(sym.ELSE, Tokens.ELSE);
  }
// fall through
case 106: break;
case 47:
  { return symbol(sym.LIST, Tokens.LIST);
  }
// fall through
case 107: break;
case 48:
  { return symbol(sym.TRUE, Tokens.TRUE);
  }
// fall through
case 108: break;
case 49:
  { return symbol(sym.BREAK, Tokens.BREAK);
  }
// fall through
case 109: break;
case 50:
  { return symbol(sym.FALSE, Tokens.FALSE);
  }
// fall through
case 110: break;
case 51:
  { return symbol(sym.PRINT, Tokens.PRINT);
  }
// fall through
case 111: break;
```

```
        case 52:
          { return symbol(sym.START, Tokens.START);
          }
        // fall through
        case 112: break;
        case 53:
          { return symbol(sym.WHILE, Tokens.WHILE);
          }
        // fall through
        case 113: break;
        case 54:
          { return symbol(sym.APPEND, Tokens.APPEND);
          }
        // fall through
        case 114: break;
        case 55:
          { return symbol(sym.DOUBLE, Tokens.DOUBLE);
          }
        // fall through
        case 115: break;
        case 56:
          { return symbol(sym.RETURN, Tokens.RETURN);
          }
        // fall through
        case 116: break;
        case 57:
          { return symbol(sym.STRING, Tokens.STRING);
          }
        // fall through
        case 117: break;
        case 58:
          { return symbol(sym.SWITCH, Tokens.SWITCH);
          }
        // fall through
        case 118: break;
        case 59:
          { return symbol(sym.DEFAULT, Tokens.DEFAULT);
          }
        // fall through
        case 119: break;
        case 60:
          { return symbol(sym.CONTINUE, Tokens.CONTINUE);
          }
        // fall through
        case 120: break;
        default:
          zzScanError(ZZ_NO_MATCH);
      }
    }
  }
 }


}
```