```
// 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;
import analizadores.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\43\1\44\1\45\1\46\1\40\1\47\1\50\1\51"+
  "\1\52\1\53\1\54\3\40\1\55\1\56\1\57\1\60"+
  "\1\61\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\43\1\44"+
  "\1\45\1\46\1\40\1\47\1\50\1\51\1\52\1\53"+
  "\1\54\3\40\1\62\1\63\1\64\7\0\1\3\252\0"+
  "\2\65\115\0\1\66\u01a8\0\2\3\u0100\0\1\67\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\4\23\1\0"+
  "\1\43\3\23\1\0\13\23\1\44\1\43\3\0\1\45"+
  "\1\0\1\46\4\23\1\0\5\23\1\0\2\23\1\47"+
```

```
  "\1\0\1\50\2\23\1\0\1\51\1\23\1\0\6\23"+
  "\1\0\1\23\1\0\1\23\1\52\1\23\1\0\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\2\60"+
  "\1\23\1\0\5\23\1\0\1\23\1\0\1\23\1\0"+
  "\1\61\2\62\1\23\2\0\1\23\2\63\1\23\1\0"+
  "\2\23\2\64\1\23\2\65\2\23\1\66\2\23\1\0"+
  "\1\23\1\0\1\23\1\0\2\67\1\0\1\70\1\23"+
  "\1\0\1\23\1\71\1\72\1\23\1\0\1\73\1\74"+
  "\1\23\2\75\2\76\1\23\2\0\1\23\1\0\1\77"+
  "\2\100\2\23\2\0\2\101\1\23\1\0\2\102\1\23"+
  "\1\0\2\103";

private static int [] zzUnpackAction() {
  int [] result = new int[238];
  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\70\0\160\0\250\0\340\0\70\0\u0118\0\u0150"+
  "\0\70\0\70\0\u0188\0\u01c0\0\70\0\u01f8\0\70\0\u0230"+
  "\0\u0268\0\70\0\70\0\u02a0\0\u02d8\0\u0310\0\u0348\0\u0380"+
  "\0\u03b8\0\u03f0\0\u0428\0\u0460\0\u0498\0\u04d0\0\u0508\0\u0540"+
  "\0\u0578\0\u05b0\0\u05e8\0\u0620\0\u0658\0\u0690\0\70\0\70"+
  "\0\70\0\70\0\u06c8\0\70\0\u0700\0\u0738\0\70\0\340"+
  "\0\70\0\u0770\0\70\0\u07a8\0\u07e0\0\70\0\70\0\70"+
  "\0\u0818\0\u0850\0\u0888\0\70\0\70\0\70\0\u08c0\0\u08f8"+
  "\0\u0930\0\u0968\0\u09a0\0\u09d8\0\u0a10\0\u0a48\0\u0a80\0\u0ab8"+
  "\0\u0af0\0\u0b28\0\u0b60\0\u0498\0\u0b98\0\u0bd0\0\u0c08\0\u0c40"+
  "\0\u0c78\0\u0cb0\0\u0ce8\0\u0d20\0\u0d58\0\u0d90\0\u0dc8\0\u0e00"+
  "\0\u0e38\0\u0e70\0\u0ea8\0\70\0\70\0\u0ee0\0\u0f18\0\u0f50"+
  "\0\70\0\u0f88\0\u0888\0\u0fc0\0\u0ff8\0\u1030\0\u1068\0\u10a0"+
  "\0\u10d8\0\u1110\0\u1148\0\u1180\0\u11b8\0\u11f0\0\u1228\0\u1260"+
  "\0\u0498\0\u1298\0\u0498\0\u12d0\0\u1308\0\u1340\0\u0498\0\u1378"+
  "\0\u13b0\0\u13e8\0\u1420\0\u1458\0\u1490\0\u14c8\0\u1500\0\u1538"+
  "\0\u1570\0\u15a8\0\u15e0\0\u0498\0\u1618\0\u1650\0\u1688\0\u16c0"+
  "\0\70\0\u16f8\0\u1730\0\u0818\0\u1768\0\u0498\0\u17a0\0\u0498"+
```

```
  "\0\70\0\u0498\0\u17d8\0\u1810\0\u1848\0\u0498\0\70\0\u1880"+
  "\0\u18b8\0\u0498\0\70\0\u18f0\0\u0498\0\70\0\u1928\0\u1960"+
  "\0\u1998\0\u19d0\0\u1a08\0\u1a40\0\u1a78\0\u1ab0\0\u1ae8\0\u1b20"+
  "\0\u1b58\0\u1b90\0\u0498\0\u0498\0\70\0\u1bc8\0\u1c00\0\u1c38"+
  "\0\u1c70\0\u0498\0\70\0\u1ca8\0\u1ce0\0\u1d18\0\u1d50\0\u0498"+
  "\0\70\0\u1d88\0\u1dc0\0\u1df8\0\u1e30\0\u1e68\0\u0498\0\u1ea0"+
  "\0\u1ed8\0\u1f10\0\u1f48\0\u1f80\0\u1fb8\0\u1ff0\0\u0498\0\70"+
  "\0\u2028\0\u0498\0\u2060\0\u2098\0\u20d0\0\u0498\0\u0498\0\u2108"+
  "\0\u2140\0\u0498\0\u0498\0\u2178\0\u0498\0\70\0\u0498\0\70"+
  "\0\u21b0\0\u21e8\0\u2220\0\u2258\0\u2290\0\u0498\0\u0498\0\70"+
  "\0\u22c8\0\u2300\0\u2338\0\u2370\0\u0498\0\70\0\u23a8\0\u23e0"+
  "\0\u0498\0\70\0\u2418\0\u2450\0\u0498\0\70";

private static int [] zzUnpackRowMap() {
  int [] result = new int[238];
  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\35\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\71\0\2\3\112\0"+
  "\1\57\42\0\5\60\1\61\50\60\1\62\11\60\7\0"+
  "\1\63\60\0\10\64\1\0\45\64\1\65\11\64\13\0"+
  "\1\66\70\0\1\67\71\0\1\70\64\0\1\71\4\0"+
  "\1\72\66\0\1\73\1\0\1\21\73\0\1\74\67\0"+
  "\1\75\67\0\1\76\63\0\1\35\5\0\17\35\1\77"+
  "\6\35\4\0\1\35\27\0\1\35\5\0\16\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\6\35\1\104\7\35\4\0\1\35"+
  "\27\0\1\35\5\0\4\35\1\105\11\35\1\106\7\35"+
  "\4\0\1\35\27\0\1\35\5\0\13\35\1\107\12\35"+
  "\4\0\1\35\27\0\1\35\5\0\1\110\7\35\1\111"+
  "\5\35\1\112\7\35\4\0\1\35\3\0\1\113\23\0"+
  "\1\35\5\0\26\35\4\0\1\35\27\0\1\35\5\0"+
```

"\5\35\1\114\7\35\1\115\10\35\4\0\1\35\27\0"+
"\1\35\5\0\4\35\1\116\3\35\1\117\15\35\4\0"+
"\1\35\3\0\1\120\23\0\1\35\5\0\4\35\1\121"+
"\21\35\4\0\1\35\27\0\1\35\5\0\20\35\1\122"+
"\5\35\4\0\1\35\27\0\1\35\5\0\4\35\1\123"+
"\11\35\1\124\7\35\4\0\1\35\27\0\1\35\5\0"+
"\22\35\1\125\2\35\1\126\4\0\1\35\27\0\1\35"+
"\5\0\16\35\1\127\1\35\1\130\5\35\4\0\1\35"+
"\27\0\1\35\5\0\1\131\15\35\1\132\7\35\4\0"+
"\1\35\27\0\1\35\5\0\7\35\1\133\16\35\4\0"+
"\1\35\71\0\1\134\40\0\1\135\7\0\1\136\74\0"+
"\1\137\2\0\1\140\13\0\2\60\2\0\64\60\10\0"+
"\1\141\57\0\2\64\2\0\64\64\13\71\1\142\54\71"+
"\2\72\2\0\64\72\21\0\1\143\67\0\1\35\5\0"+
"\17\35\1\144\6\35\4\0\1\35\27\0\1\35\5\0"+
"\16\35\1\145\7\35\4\0\1\35\27\0\1\35\5\0"+
"\4\35\1\146\21\35\4\0\1\35\27\0\1\35\5\0"+
"\21\35\1\147\4\35\4\0\1\35\4\0\1\150\22\0"+
"\1\35\5\0\1\151\25\35\4\0\1\35\27\0\1\35"+
"\5\0\15\35\1\152\10\35\4\0\1\35\27\0\1\35"+
"\5\0\5\35\1\153\20\35\4\0\1\35\27\0\1\35"+
"\5\0\23\35\1\154\2\35\4\0\1\35\27\0\1\35"+
"\5\0\21\35\1\155\4\35\4\0\1\35\4\0\1\156"+
"\22\0\1\35\5\0\13\35\1\157\12\35\4\0\1\35"+
"\27\0\1\35\5\0\15\35\1\160\10\35\4\0\1\35"+
"\27\0\1\35\5\0\20\35\1\161\5\35\4\0\1\35"+
"\52\0\1\162\44\0\1\35\5\0\22\35\1\163\3\35"+
"\4\0\1\35\27\0\1\35\5\0\15\35\1\164\10\35"+
"\4\0\1\35\27\0\1\35\5\0\21\35\1\165\4\35"+
"\4\0\1\35\4\0\1\166\51\0\1\166\15\0\1\166"+
"\22\0\1\35\5\0\25\35\1\167\4\0\1\35\27\0"+
"\1\35\5\0\10\35\1\170\15\35\4\0\1\35\3\0"+
"\1\171\23\0\1\35\5\0\14\35\1\172\5\35\1\173"+
"\3\35\4\0\1\35\27\0\1\35\5\0\23\35\1\174"+
"\2\35\4\0\1\35\27\0\1\35\5\0\1\175\17\35"+
"\1\176\5\35\4\0\1\35\27\0\1\35\5\0\10\35"+
"\1\177\15\35\4\0\1\35\3\0\1\200\23\0\1\35"+
"\5\0\21\35\1\201\4\35\4\0\1\35\4\0\1\202"+
"\22\0\1\35\5\0\23\35\1\203\2\35\4\0\1\35"+
"\27\0\1\35\5\0\20\35\1\204\5\35\4\0\1\35"+
"\27\0\1\35\5\0\10\35\1\205\15\35\4\0\1\35"+
"\3\0\1\206\23\0\1\35\5\0\10\35\1\207\15\35"+
"\4\0\1\35\3\0\1\210\53\0\1\211\45\0\1\212"+
"\17\0\1\213\57\0\1\200\25\0\1\200\2\0\13\71"+
"\1\142\4\71\1\214\47\71\21\0\1\35\5\0\4\35"+
"\1\215\21\35\4\0\1\35\27\0\1\35\5\0\13\35"+
"\1\216\12\35\4\0\1\35\27\0\1\35\5\0\1\217"+
"\25\35\4\0\1\35\27\0\1\35\5\0\4\35\1\220"+
"\21\35\4\0\1\35\41\0\1\221\55\0\1\35\5\0"+
"\20\35\1\222\5\35\4\0\1\35\27\0\1\35\5\0"+
"\22\35\1\223\3\35\4\0\1\35\27\0\1\35\5\0"+
"\1\224\25\35\4\0\1\35\27\0\1\35\5\0\1\35"+
"\1\225\24\35\4\0\1\35\27\0\1\35\5\0\4\35"+
"\1\226\21\35\4\0\1\35\41\0\1\227\55\0\1\35"+
"\5\0\21\35\1\230\4\35\4\0\1\35\4\0\1\231"+
"\22\0\1\35\5\0\3\35\1\232\22\35\4\0\1\35"+
"\40\0\1\233\56\0\1\35\5\0\6\35\1\234\17\35"+

```
"\4\0\1\35\27\0\1\35\5\0\22\35\1\235\3\35"+
"\4\0\1\35\57\0\1\236\37\0\1\35\5\0\15\35"+
"\1\237\10\35\4\0\1\35\52\0\1\240\44\0\1\35"+
"\5\0\16\35\1\241\7\35\4\0\1\35\27\0\1\35"+
"\5\0\23\35\1\242\2\35\4\0\1\35\27\0\1\35"+
"\5\0\15\35\1\243\10\35\4\0\1\35\27\0\1\35"+
"\5\0\20\35\1\244\5\35\4\0\1\35\27\0\1\35"+
"\5\0\10\35\1\245\15\35\4\0\1\35\3\0\1\246"+
"\23\0\1\35\5\0\22\35\1\247\3\35\4\0\1\35"+
"\57\0\1\250\37\0\1\35\5\0\22\35\1\251\3\35"+
"\4\0\1\35\57\0\1\252\37\0\1\35\5\0\4\35"+
"\1\253\21\35\4\0\1\35\27\0\1\35\5\0\3\35"+
"\1\254\22\35\4\0\1\35\40\0\1\255\56\0\1\35"+
"\5\0\13\35\1\256\12\35\4\0\1\35\50\0\1\257"+
"\74\0\1\260\57\0\1\246\25\0\1\246\23\0\1\35"+
"\5\0\15\35\1\261\10\35\4\0\1\35\27\0\1\35"+
"\5\0\12\35\1\262\13\35\4\0\1\35\5\0\1\263"+
"\21\0\1\35\5\0\10\35\1\264\15\35\4\0\1\35"+
"\3\0\1\265\23\0\1\35\5\0\23\35\1\266\2\35"+
"\4\0\1\35\27\0\1\35\5\0\13\35\1\267\12\35"+
"\4\0\1\35\27\0\1\35\5\0\4\35\1\270\21\35"+
"\4\0\1\35\41\0\1\271\55\0\1\35\5\0\22\35"+
"\1\272\3\35\4\0\1\35\27\0\1\35\5\0\22\35"+
"\1\273\3\35\4\0\1\35\57\0\1\274\37\0\1\35"+
"\5\0\24\35\1\275\1\35\4\0\1\35\27\0\1\35"+
"\5\0\20\35\1\276\5\35\4\0\1\35\27\0\1\35"+
"\5\0\3\35\1\277\22\35\4\0\1\35\27\0\1\35"+
"\5\0\22\35\1\300\3\35\4\0\1\35\27\0\1\35"+
"\5\0\15\35\1\301\10\35\4\0\1\35\52\0\1\302"+
"\44\0\1\35\5\0\2\35\1\303\23\35\4\0\1\35"+
"\37\0\1\304\57\0\1\35\5\0\20\35\1\305\5\35"+
"\4\0\1\35\55\0\1\306\41\0\1\35\5\0\4\35"+
"\1\307\21\35\4\0\1\35\41\0\1\310\105\0\1\311"+
"\37\0\1\35\5\0\3\35\1\312\22\35\4\0\1\35"+
"\27\0\1\35\5\0\15\35\1\313\10\35\4\0\1\35"+
"\52\0\1\314\44\0\1\35\5\0\13\35\1\315\12\35"+
"\4\0\1\35\27\0\1\35\5\0\4\35\1\316\21\35"+
"\4\0\1\35\27\0\1\35\5\0\7\35\1\317\16\35"+
"\4\0\1\35\27\0\1\35\5\0\13\35\1\320\12\35"+
"\4\0\1\35\50\0\1\321\46\0\1\35\5\0\4\35"+
"\1\322\21\35\4\0\1\35\27\0\1\35\5\0\15\35"+
"\1\323\10\35\4\0\1\35\27\0\1\35\5\0\26\35"+
"\4\0\1\324\27\0\1\35\5\0\6\35\1\325\17\35"+
"\4\0\1\35\43\0\1\326\53\0\1\35\5\0\7\35"+
"\1\327\16\35\4\0\1\35\44\0\1\330\52\0\1\35"+
"\5\0\10\35\1\331\15\35\4\0\1\35\3\0\1\332"+
"\41\0\1\332\25\0\1\332\63\0\1\333\27\0\1\35"+
"\5\0\23\35\1\334\2\35\4\0\1\35\60\0\1\335"+
"\36\0\1\35\5\0\22\35\1\336\3\35\4\0\1\35"+
"\27\0\1\35\5\0\15\35\1\337\10\35\4\0\1\35"+
"\52\0\1\340\44\0\1\35\5\0\25\35\1\341\4\0"+
"\1\35\27\0\1\35\5\0\15\35\1\342\10\35\4\0"+
"\1\35\52\0\1\343\77\0\1\344\34\0\1\35\5\0"+
"\4\35\1\345\21\35\4\0\1\35\41\0\1\346\55\0"+
"\1\35\5\0\10\35\1\347\15\35\4\0\1\35\3\0"+
"\1\350\23\0\1\35\5\0\6\35\1\351\17\35\4\0"+
"\1\35\43\0\1\352\71\0\1\350\25\0\1\350\23\0"+
```

```
  "\1\35\5\0\22\35\1\353\3\35\4\0\1\35\57\0"+
  "\1\354\37\0\1\35\5\0\7\35\1\355\16\35\4\0"+
  "\1\35\44\0\1\356\31\0";

private static int [] zzUnpacktrans() {
  int [] result = new int[9352];
  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\14\1\1\0\4\1\1\0"+
  "\13\1\2\11\3\0\1\11\1\0\5\1\1\0\5\1"+
  "\1\0\3\1\1\0\3\1\1\0\2\1\1\0\6\1"+
  "\1\0\1\1\1\0\3\1\1\0\1\1\1\0\1\11"+
  "\2\0\5\1\1\11\5\1\1\11\1\1\1\0\1\1"+
  "\1\11\2\1\1\11\1\1\1\0\5\1\1\0\1\1"+
```

```
  "\1\0\1\1\1\0\2\1\1\11\1\1\2\0\2\1"+
  "\1\11\1\1\1\0\3\1\1\11\10\1\1\0\1\1"+
  "\1\0\1\1\1\0\1\1\1\11\1\0\2\1\1\0"+
  "\4\1\1\0\4\1\1\11\1\1\1\11\1\1\2\0"+
  "\1\1\1\0\2\1\1\11\2\1\2\0\1\1\1\11"+
  "\1\1\1\0\1\1\1\11\1\1\1\0\1\1\1\11";

private static int [] zzUnpackAttribute() {
  int [] result = new int[238];
  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: */
  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);
  }

  private Symbol symbol(int type, Tokens tokenTipo) {
     TokenLista.guardar(tokenTipo, yytext(), yyline+1, yycolumn+1);
     return new Symbol(type, yyline+1, yycolumn+1, yytext());
  }


/**
 * 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);
System.err.println("Error léxico: " + yytext() + " en línea " + (yyline+1));
        }
      // fall through
      case 68: break;
      case 2:
        { /* Ignorar */
        }
      // fall through
      case 69: break;
      case 3:
        { return symbol(sym.NOT, Tokens.NOT);
        }
      // fall through
      case 70: break;
      case 4:
        { return symbol(sym.MOD, Tokens.MOD);
        }
      // fall through
      case 71: break;
      case 5:
        { return symbol(sym.PAR_A, Tokens.PAR_A);
        }
      // fall through
      case 72: break;
      case 6:
        { return symbol(sym.PAR_C, Tokens.PAR_C);
        }
      // fall through
      case 73: break;
      case 7:
        { return symbol(sym.POR, Tokens.MULT);
        }
      // fall through
      case 74: break;
      case 8:
        { return symbol(sym.MAS, Tokens.SUMA);
        }
      // fall through
      case 75: break;
      case 9:
        { return symbol(sym.COMA, Tokens.COMA);
        }
      // fall through
      case 76: break;
      case 10:
        { return symbol(sym.MENOS, Tokens.RESTA);
```

```
 }
// fall through
case 77: break;
case 11:
 { return symbol(sym.PUNTO, Tokens.PUNTO);
 }
// fall through
case 78: break;
case 12:
 { return symbol(sym.DIV, Tokens.DIV);
 }
// fall through
case 79: break;
case 13:
 { return symbol(sym.ENTERO, yytext(), Tokens.ENTERO);
 }
// fall through
case 80: break;
case 14:
 { return symbol(sym.DOSPTOS, Tokens.DOS_PUNTOS);
 }
// fall through
case 81: break;
case 15:
 { return symbol(sym.PTCOMA, Tokens.PUNTO_COMA);
 }
// fall through
case 82: break;
case 16:
 { return symbol(sym.MENOR, Tokens.MENOR);
 }
// fall through
case 83: break;
case 17:
 { return symbol(sym.IGUAL, Tokens.ASIGN);
 }
// fall through
case 84: break;
case 18:
 { return symbol(sym.MAYOR, Tokens.MAYOR);
 }
// fall through
case 85: break;
case 19:
 { return symbol(sym.ID, yytext(), Tokens.ID);
 }
// fall through
case 86: break;
case 20:
 { return symbol(sym.COR_IZQ, Tokens.COR_IZQ);
 }
// fall through
case 87: break;
case 21:
 { return symbol(sym.COR_DER, Tokens.COR_DER);
 }
// fall through
```

```
 case 88: break;
 case 22:
  { return symbol(sym.XOR, Tokens.XOR);
  }
 // fall through
 case 89: break;
 case 23:
  { return symbol(sym.LLAVE_A, Tokens.LLAVE_A);
  }
 // fall through
 case 90: break;
 case 24:
  { return symbol(sym.LLAVE_C, Tokens.LLAVE_C);
  }
 // fall through
 case 91: break;
 case 25:
  { return symbol(sym.DIFERENTE, Tokens.DIF);
  }
 // fall through
 case 92: break;
 case 26:
  { String val = yytext();
String contenido = val.substring(1, val.length()-1);
// Procesar secuencias de escape
contenido = contenido.replace("\\n", "\n")
                .replace("\\t", "\t")
                .replace("\\r", "\r")
                .replace("\\'", "'")
                .replace("\\\"", "\"")
                .replace("\\\\", "\\");
TokenLista.guardar(Tokens.LIT_STRING, val, yyline+1, yycolumn+1);
return new Symbol(sym.LIT_STRING, yyline+1, yycolumn+1, contenido);
  }
 // fall through
 case 93: break;
 case 27:
  { return symbol(sym.AND, Tokens.AND);
  }
 // fall through
 case 94: break;
 case 28:
  { return symbol(sym.POTENCIA, Tokens.POT);
  }
 // fall through
 case 95: break;
 case 29:
  { return symbol(sym.MASMAS, Tokens.INCREMENTO);
  }
 // fall through
 case 96: break;
 case 30:
  { return symbol(sym.MENOSMENOS, Tokens.DECREMENTO);
  }
 // fall through
 case 97: break;
 case 31:
```

```
  { return symbol(sym.MENORIGUAL, Tokens.MENORIGUAL);
  }
 // fall through
 case 98: break;
 case 32:
  { return symbol(sym.IGUALIGUAL, Tokens.IGUAL);
  }
 // fall through
 case 99: break;
 case 33:
  { return symbol(sym.MAYORIGUAL, Tokens.MAYORIGUAL);
  }
 // fall through
 case 100: break;
 case 34:
  { return symbol(sym.DO, Tokens.DO);
  }
 // fall through
 case 101: break;
 case 35:
  { return symbol(sym.IF, Tokens.IF);
  }
 // fall through
 case 102: break;
 case 36:
  { return symbol(sym.OR, Tokens.OR);
  }
 // fall through
 case 103: break;
 case 37:
  { String val = yytext();
String contenido = val.substring(1, val.length()-1);
// Procesar secuencias de escape
contenido = contenido.replace("\\n", "\n")
                .replace("\\t", "\t")
                .replace("\\'", "'")
                .replace("\\\\", "\\");
TokenLista.guardar(Tokens.LIT_CHAR, val, yyline+1, yycolumn+1);
return new Symbol(sym.LIT_CHAR, yyline+1, yycolumn+1, contenido);
  }
 // fall through
 case 104: break;
 case 38:
  { return symbol(sym.DECIMAL, yytext(), Tokens.DECIMAL);
  }
 // fall through
 case 105: break;
 case 39:
  { return symbol(sym.FOR, Tokens.FOR);
  }
 // fall through
 case 106: break;
 case 40:
  { return symbol(sym.INT, Tokens.INT);
  }
 // fall through
 case 107: break;
```

```
case 41:
  { return symbol(sym.NEW, Tokens.NEW);
  }
// fall through
case 108: break;
case 42:
  { return symbol(sym.VAR, Tokens.VAR);
  }
// fall through
case 109: break;
case 43:
  { return symbol(sym.BOOL, Tokens.BOOL);
  }
// fall through
case 110: break;
case 44:
  { return symbol(sym.CASE, Tokens.CASE);
  }
// fall through
case 111: break;
case 45:
  { return symbol(sym.CHAR, Tokens.CHAR);
  }
// fall through
case 112: break;
case 46:
  { return symbol(sym.ELSE, Tokens.ELSE);
  }
// fall through
case 113: break;
case 47:
  { return symbol(sym.FIND, Tokens.FIND);
  }
// fall through
case 114: break;
case 48:
  { return symbol(sym.LIST, Tokens.LIST);
  }
// fall through
case 115: break;
case 49:
  { return symbol(sym.TRUE, "true", Tokens.TRUE);
  }
// fall through
case 116: break;
case 50:
  { return symbol(sym.VOID, Tokens.VOID);
  }
// fall through
case 117: break;
case 51:
  { return symbol(sym.BREAK, Tokens.BREAK);
  }
// fall through
case 118: break;
case 52:
  { return symbol(sym.FALSE, "false", Tokens.FALSE);
```

```
 }
// fall through
case 119: break;
case 53:
 { return symbol(sym.PRINT, Tokens.PRINT);
 }
// fall through
case 120: break;
case 54:
 { return symbol(sym.ROUND, Tokens.ROUND);
 }
// fall through
case 121: break;
case 55:
 { return symbol(sym.WHILE, Tokens.WHILE);
 }
// fall through
case 122: break;
case 56:
 { return symbol(sym.APPEND, Tokens.APPEND);
 }
// fall through
case 123: break;
case 57:
 { return symbol(sym.DOUBLE, Tokens.DOUBLE);
 }
// fall through
case 124: break;
case 58:
 { return symbol(sym.LENGTH, Tokens.LENGTH);
 }
// fall through
case 125: break;
case 59:
 { return symbol(sym.REMOVE, Tokens.REMOVE);
 }
// fall through
case 126: break;
case 60:
 { return symbol(sym.RETURN, Tokens.RETURN);
 }
// fall through
case 127: break;
case 61:
 { return symbol(sym.STRING, Tokens.STRING);
 }
// fall through
case 128: break;
case 62:
 { return symbol(sym.SWITCH, Tokens.SWITCH);
 }
// fall through
case 129: break;
case 63:
 { return symbol(sym.DEFAULT, Tokens.DEFAULT);
 }
// fall through
```

```
        case 130: break;
        case 64:
          { return symbol(sym.PRINTLN, Tokens.PRINTLN);
          }
        // fall through
        case 131: break;
        case 65:
          { return symbol(sym.CONTINUE, Tokens.CONTINUE);
          }
        // fall through
        case 132: break;
        case 66:
          { return symbol(sym.TOSTRING, Tokens.TOSTRING);
          }
        // fall through
        case 133: break;
        case 67:
          { return symbol(sym.START_WITH, Tokens.START_WITH);
          }
        // fall through
        case 134: break;
        default:
          zzScanError(ZZ_NO_MATCH);
      }
    }
  }
 }


}
```