Nombre: **Trabajo Final Compiladores**
Carrera: **Lic. en Sistemas Informáticos>**
Materia: **Diseño de Compiladores**
Cátedra: **Ing. Pablo Pandolfo**
Año de Curso: **2015** Turno: **Noche**
Entrega: Final
Fecha requerida: **2015-11-12** Fecha entregada: **2015-11-12**

Grupo

ID/Matrícula	APELLIDO, Nombres	Correo Electrónico
128706	Freijo, Joaquin	joaquin.freijo@comunidad.ub.edu.ar
11155	Melllace, Gabriel	gabriel.mellace@comunidad.ub.edu.ar

Grilla de calificación

Concepto	Funcionalidad	Contenidos	Resolución	Formato	Ampliaciones
Sobresaliente (10)					
Distinguido (9-8)					
Bueno (7-6)					
Aprobado (5-4)					
Insuficiente (3-2-1)					
Reprobado (0)					
NOTA					

Comentario adicional del Profesor:

Descripción:
RobotGrid [basado en el lenguaje Karel] es un lenguaje de programación sencillo que tiene como objetivo enseñar conceptos básicos de programación de una forma gráfica. Mediante instrucciones, se mueve a un robot en una grilla rectangular. Cada espacio puede estar libre o ser una pared.

El programa consiste de dos secciones:
- Inicializaci'on de la grilla y el robot

Se genera la grilla especificando su tamaño (máximo 9 x 9)
Se especifica donde se encuentran las paredes
Se ubica al robot (si hay una pared en ese lugar se sobreescribe) y su orientación

- Acciones a ejecutar (delimitadas por START y END)
El lenguaje cuenta con dos instrucciones:
turnLeft : gira el robot en sentido contrario a las agujas del reloj
move: mueve el robot una casilla en la dirección en la que se encuentra

Tiene 3 sentencias de control:
iterate (NUM) : similar a un for, realiza la acción el numero especificado en NUM
while (COND): mientras la condicion sea cierta, se ejecuta la accion
if (COND) : realiza la acción si la condición es cierta

Condiciones:
frontClear : si la celda adelante del robot esta libre, evalua a true

Errores de Ejecucion:
- Si el robot choca contra una pared o se va de los limites de grilla, el programa imprime un GAME OVER e ignora las instrucciones siguientes

Objetivo:
- Mediante las herramientas JFlex y Cup, se genera un compilador que toma instrucciones del lenguaje Karel y ejecuta acciones de Lenguaje Java.

Alcance:
- No hay manejo ni recuperación de errores: si se da como entrada un programa ilegal, la compilación del programa fallara con el error default de Cup.

Requerimientos:
Java 1.8
Jflex 1.6.1
Cup 0.11a

JFLEX

```
package compilador.modelos;
import java_cup.runtime.Symbol;

%%

%public
%class AnalizadorLexico
%standalone
%cup

%%

// Directions
N | S | E | W     { return new Symbol(sym.DIR, yytext()); }


// Control Statements
if                      { return new Symbol(sym.IF); }
while             { return new Symbol(sym.WHILE); }
iterate                 { return new Symbol(sym.ITERATE); }
\d                      { return new Symbol(sym.NUMBER,
Integer.valueOf(yytext())); }

// Start/End
START                   { return new Symbol(sym.START); }
END                     { return new Symbol(sym.END); }

// Initialization statements
drawGrid          { return new Symbol(sym.DRAWGRID); }
placeRobot        { return new Symbol(sym.PLACEROBOT); }
placeWall         { return new Symbol(sym.PLACEWALL); }

// Built-in instructions
move              { return new Symbol(sym.MOVE); }
turnLeft          { return new Symbol(sym.TURNLEFT); }

// Conditions
frontClear        { return new Symbol(sym.FCLEAR); }

// Other
"("                     { return new Symbol(sym.LPAREN); }
")"                     { return new Symbol(sym.RPAREN); }
,                       { return new Symbol(sym.COMMA); }
;                       { return new Symbol(sym.SCOLON); }
"{"                     { return new Symbol(sym.LBRACE); }
```

```
"}"                     { return new Symbol(sym.RBRACE); }
"!"                     { return new Symbol(sym.NEQ); }
\s                      {  } //ignorar whitespace
.               { System.out.print(yytext() + "[Caracter Invalido]");
}
```

CUP

```
package compilador.modelos;
import java_cup.runtime.*;

action code
{:

      public void forFunc (int num, int instr) {
            for (int i = 0; i < num; i++) {
                  if (instr==0) GridMap.move();
                  else if (instr==1) GridMap.turnLeft();
                  else throw new IllegalArgumentException("forFunc:
Illegal Instruction");
                  }

 :}


terminal
      IF, WHILE, ITERATE, START, END,
      DRAWGRID, PLACEROBOT, PLACEWALL, MOVE, TURNLEFT, FCLEAR,
      LPAREN, RPAREN, COMMA, SCOLON, RBRACE, LBRACE, NEQ;
terminal String DIR;
terminal Integer NUMBER;

non terminal
      program, initial, robot, wallList, wall,
      statementList, statement, iteration, loop, conditional;

non terminal Integer instruction, cond;
non terminal GridMap grid;

program ::= initial START statementList END  {:
System.out.println("[Parse completed succesfully]"); :}
            ;

initial ::= grid wallList robot
                        {: GridMap.printState(); :}
                  ;

grid ::= DRAWGRID LPAREN NUMBER:n1 COMMA NUMBER:n2 RPAREN SCOLON
                  {: RESULT = new GridMap(n1,n2);  :}
            ;
```

```
wallList ::= wallList wall
                  | //empty
                  ;

wall ::= PLACEWALL LPAREN NUMBER:n1 COMMA NUMBER:n2 RPAREN SCOLON
                  {:  GridMap.setElement(n1,n2,"Wall");  :}
            ;

robot ::= PLACEROBOT LPAREN NUMBER:n1 COMMA NUMBER:n2 COMMA DIR:d
RPAREN SCOLON
                  {: GridMap.setElement(n1,n2,d.charAt(0)); :}
            ;

statementList ::= statementList statement
                  |     statement
                  ;

statement ::= iteration
                  | loop
                  | conditional
                  | instruction:i {: if (i == 0) GridMap.move(); else
GridMap.turnLeft(); :}
                  ;

iteration ::= ITERATE LPAREN NUMBER:n RPAREN LBRACE instruction:i
RBRACE
                  {:  forFunc(n, i); :}
                  ;

conditional ::= IF LPAREN cond:b RPAREN LBRACE instruction:i RBRACE
                  {:     if (b == 0) {
                                    if (GridMap.isFrontClear()) { if
(i == 0) GridMap.move(); else GridMap.turnLeft(); }
                                    }
                              else { if (!GridMap.isFrontClear()) { if (i ==
0) GridMap.move(); else GridMap.turnLeft(); }
                              }
                  :}
                  ;

loop ::= WHILE LPAREN cond:b RPAREN LBRACE instruction:i RBRACE
                  {:     if (b == 0) {
                                    while (GridMap.isFrontClear()) {
if (i == 0) GridMap.move(); else GridMap.turnLeft(); }
                                    }
                              else { while (!GridMap.isFrontClear()) { if (i
== 0) GridMap.move(); else GridMap.turnLeft(); }
                              }
                  :}
```

```
                ;

cond ::= FCLEAR   {: RESULT = 0; :}
           | NEQ FCLEAR {: RESULT = 1; :}
           ;

instruction ::= MOVE SCOLON {: RESULT = 0; :}
                | TURNLEFT SCOLON {: RESULT = 1; :}
                ;
```

GridMap.java

```
/* 0 -> Robot (North)
* 1 -> Robot (West)
* 2 -> Robot (South)
* 3 -> Robot (East)
* 4 -> Empty
* 5 -> Wall
* 6 -> Beeper
* 7 -> Dead
*/

package compilador.modelos;

public class GridMap {
      private static int nRows; // height
      private static int nColumns; // width
      private static int [] grid;
      private static int robotPos;
      private static int robotStatus; // 0 alive, 1 dead

      public GridMap(int rows, int cols) {
            assert (rows > 0 && cols > 0) : "Grid can't be empty" ;
                  nRows = rows;
                  nColumns = cols;
                  grid = new int [rows*cols];
                  robotStatus = 0;


                  // populate grid with empty cells
                  for (int i=0; i < grid.length; i++) {
                              grid[i] = 4;
                  }

      }

      public static int getHeight() {
            return nRows;
```

```
	}


	public static int getWidth() {
		return nColumns;
	}

	public static int getRobotPos() {
		return robotPos;
	}

	public static char getRobotDirection() {
		char dir = ' ';
		switch (grid[robotPos])
		{
		case 0: dir = 'N'; break;
		case 1: dir = 'W'; break;
		case 2: dir = 'S'; break;
		case 3: dir = 'E'; break;
		default: throw new IllegalArgumentException("Invalid
element: " + grid[robotPos]);
		}

		return dir;

	}


	public static void setElement(int row, int col, String element)
{
		int position = nColumns * row + col; // map 2D into 1D,
row order

		try {
			switch(element) {
			case "North" : grid[position] = 0; robotPos =
position; break;
			case "West" : grid[position] = 1; robotPos =
position; break;
			case "South" : grid[position] = 2; robotPos =
position; break;
			case "East" : grid[position] = 3; robotPos =
position; break;
			case "Empty" : grid[position] = 4; break;
			case "Wall" : grid[position] = 5; break;
			case "Beeper" : grid[position] = 6; break;
			default : System.out.println("Invalid element: " +
element);

			}
```

```
            } catch (Exception e) { System.out.println("Position [" +
row + "," + col + "] Out of Bounds" ); }

      }

      public static void setElement(int row, int col, char element) {
            int position = nColumns * row + col; // map 2D into 1D,
row order

            try {
                  switch(element) {
                  case 'N' : grid[position] = 0; robotPos = position;
break;
                  case 'W' : grid[position] = 1; robotPos = position;
break;
                  case 'S' : grid[position] = 2; robotPos = position;
break;
                  case 'E' : grid[position] = 3; robotPos = position;
break;
                  case 'O' : grid[position] = 4; break;
                  case 'X' : grid[position] = 5; break;
                  case 'B' : grid[position] = 6; break;
                  default : System.out.println("Invalid element: " +
element);
                  }
            } catch (Exception e) { System.out.println("Position [" +
row + "," + col + "] Out of Bounds" ); }

      }


      public static String getElement(int row, int col) {
            String element = ""; // map 2D into 1D, row order

            try {
                  switch(grid[nColumns * row + col]) {
                  case 0 : element = "North"; break;
                  case 1 : element = "West"; break;
                  case 2 : element = "South"; break;
                  case 3 : element = "East"; break;
                  case 4 : element = "Empty"; break;
                  case 5 : element = "Wall"; break;
                  case 6 : element = "Beeper"; break;
                  default : System.out.println("THIS SHOULD NEVER BE
PRINTED");
                  }
            } catch (Exception e) { System.out.println("Position [" +
row + "," + col + "] Out of Bounds" ); }

            return element;
```

```
	}


	public static boolean isCellBlocked(int row, int col) {
		if (grid[nColumns * row + col] == 5) return true;
		else return false;
	}

	public static boolean isFrontClear() {
		boolean clear = false;

		switch (grid[robotPos]) { // check direction of the robot
before determining if front is clear
		case 0 : // North
			if ( (getRobotRow() > 0) && !(grid[robotPos-
nColumns] == 5) )  clear = true;
			else clear = false;
		break;

		case 1 : // West
			if ( (getRobotCol() > 0) && !(grid[robotPos-1] == 5)
)  clear = true;
			else  clear = false;
		break;

		case 2 : // South
			if ( (getRobotRow() < nRows-1) &&
!(grid[robotPos+nColumns] == 5) ) clear = true;
			else clear = false;
		break;

		case 3 : // East
			if ( (getRobotCol() < nColumns-1) &&
!(grid[robotPos+1] == 5) ) clear = true;
			else clear = false;
		break;

		case 7 : break; // robot is dead, do nothing

		default: throw new IllegalArgumentException("frontClear:
Invalid direction");
		}

		return clear;
	}

	public static boolean isRobotAlive() {
		if  (robotStatus == 0) return true;
		else return false;
	}
```

```
	public static void turnLeft() {
		if (robotStatus==0) {
			switch (grid[robotPos]) {
			case 0 : grid[robotPos] = 1; break;
			case 1 : grid[robotPos] = 2; break;
			case 2 : grid[robotPos] = 3; break;
			case 3 : grid[robotPos] = 0; break;
			default: throw new
IllegalArgumentException("TurnLeft: Invalid direction");
			}
			printState();
		}
	}

	public static void move() {
		if (robotStatus==0) {
			if ( isFrontClear() ) {
				switch (grid[robotPos]) { // check direction
of the robot before moving
				case 0 : grid[robotPos] = 4;  robotPos -
= nColumns; grid[robotPos] = 0; break;
				case 1 : grid[robotPos] = 4;  robotPos--
; grid[robotPos] = 1; break;
				case 2 : grid[robotPos] = 4;  robotPos
+= nColumns; grid[robotPos] = 2; break;
				case 3 : grid[robotPos] = 4;
robotPos++; grid[robotPos] = 3; break;
				default: throw new
IllegalArgumentException("Move: Invalid direction");
				}
			} else {
				grid[robotPos] = 7; robotStatus = 1;
System.out.println("You hit a Wall, GAME OVER");
			}
			printState();
		}


	private static int getRobotRow() {
		return robotPos / nColumns;
	}

	private static int getRobotCol() {
		return robotPos % nColumns;
	}


	private static char intToChar(int element) {
		char print = ' ';
```

```
            switch(element) {
            case 0 : print = '^'; break;
            case 1 : print = '<'; break;
            case 2 : print = 'v'; break;
            case 3 : print = '>'; break;
            case 4 : print = ' '; break;
            case 5 : print = 'W'; break;
            case 6 : print = 'B'; break;
            case 7 : print = 'X'; break;
            default: throw new IllegalArgumentException("Invalid
element: " + element);
            }
            return print;
      }


      public static void printState() {
            System.out.println();

            for (int i=0; i < nRows; i++) {
                  for (int j=0; j < nColumns; j++) {
                        System.out.print("[" + intToChar(grid[nColumns
* i + j]) + "] ");
                        }
                  System.out.println();
                  }

            System.out.println();
            }
}
```

AnalizadorLexico.java

```
/* The following code was generated by JFlex 1.6.1 */

// http://mormegil.wz.cz/prog/karel/prog_doc.htm

package compilador.modelos;
import java_cup.runtime.Symbol;


/**
 * This class is a scanner generated by
 * <a href="http://www.jflex.de/">JFlex</a> 1.6.1
 * from the specification file
<tt>src/compilador/modelos/Lexico.flex</tt>
 */
public class AnalizadorLexico 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
  };

  /**
   * Translates characters to character classes
   */
  private static final String ZZ_CMAP_PACKED =
    "\11\0\1\46\1\46\1\46\1\46\1\46\22\0\1\46\1\45\6\0"+
    "\1\37\1\40\2\0\1\41\3\0\12\16\1\0\1\42\5\0\1\20"+
    "\1\0\1\36\1\22\1\3\1\0\1\24\4\0\1\35\1\0\1\1"+
    "\3\0\1\21\1\2\1\17\2\0\1\4\11\0\1\15\1\30\1\26"+
    "\1\23\1\12\1\6\1\0\1\10\1\5\2\0\1\11\1\31\1\34"+
    "\1\27\1\25\1\0\1\14\1\0\1\13\1\33\1\32\1\7\3\0"+
    "\1\43\1\0\1\44\7\0\1\46\32\0\1\46\u05bf\0\12\16\206\0"+
    "\12\16\306\0\12\16\u019c\0\12\16\166\0\12\16\166\0\12\16\166\0"+
    "\12\16\166\0\12\16\166\0\12\16\166\0\12\16\166\0\12\16\166\0"+
    "\12\16\166\0\12\16\140\0\12\16\166\0\12\16\106\0\12\16\u0116\0"+

"\12\16\106\0\12\16\u05e6\0\1\46\u015f\0\12\16\46\0\12\16\u012c\0"+
    "\12\16\200\0\12\16\246\0\12\16\6\0\12\16\266\0\12\16\126\0"+
    "\12\16\206\0\12\16\6\0\12\16\u03a6\0\13\46\35\0\1\46\1\46"+
    "\5\0\1\46\57\0\1\46\u0fa0\0\1\46\u761f\0\12\16\u02a6\0\12\16"+
    "\46\0\12\16\306\0\12\16\26\0\12\16\126\0\12\16\u0196\0\12\16"+

"\u5316\0\12\16\u0586\0\12\16\u0bbc\0\12\16\200\0\12\16\74\0\12\16"+

"\220\0\12\16\u0116\0\12\16\u01d6\0\12\16\u0176\0\12\16\146\0\12\16"+

"\u0216\0\12\16\u5176\0\12\16\346\0\12\16\u6c74\0\62\16\uffff\0\uffff\
0\uffff\0\uffff\0\uffff\0\uffff\0\uffff\0\uffff\0\uffff\0\uffff\0\ufff
f\0\uffff\0\uffff\0\uffff\0\uffff\0\u280f\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\3\2\4\1\1\3\3\1\1\4\1\5"+
    "\1\6\1\7\1\10\1\11\1\12\1\13\2\0\1\14"+
    "\10\0\1\15\16\0\1\16\1\17\2\0\1\20\11\0"+
    "\1\21\6\0\1\22\1\23\3\0\1\24\1\0\1\25"+
    "\1\26";

  private static int [] zzUnpackAction() {
    int [] result = new int[77];
    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\47\0\47\0\116\0\165\0\234\0\303\0\352"+
    "\0\u0111\0\47\0\u0138\0\u015f\0\u0186\0\47\0\47\0\47"+
    "\0\47\0\47\0\47\0\47\0\47\0\u01ad\0\u01d4\0\47"+

"\0\u01fb\0\u0222\0\u0249\0\u0270\0\u0297\0\u02be\0\u02e5\0\u030c"+
    "\0\47\0\u0333\0\u035a\0\u0381\0\u03a8\0\u03cf\0\u03f6\0\u041d"+
    "\0\u0444\0\u046b\0\u0492\0\u04b9\0\u04e0\0\u0507\0\u052e\0\47"+
    "\0\47\0\u0555\0\u057c\0\47\0\u05a3\0\u05ca\0\u05f1\0\u0618"+
    "\0\u063f\0\u0666\0\u068d\0\u06b4\0\u06db\0\47\0\u0702\0\u0729"+
    "\0\u0750\0\u0777\0\u079e\0\u07c5\0\47\0\47\0\u07ec\0\u0813"+
    "\0\u083a\0\47\0\u0861\0\47\0\47";

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

  private static int [] zzUnpackTrans() {
    int [] result = new int[2184];
    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[] = {
    "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</code>
   */
  private static final int [] ZZ_ATTRIBUTE = zzUnpackAttribute();

  private static final String ZZ_ATTRIBUTE_PACKED_0 =
    "\1\0\2\11\6\1\1\11\3\1\10\11\2\0\1\11"+
    "\10\0\1\11\16\0\2\11\2\0\1\11\11\0\1\11"+
    "\6\0\2\11\3\0\1\11\1\0\2\11";

  private static int [] zzUnpackAttribute() {
    int [] result = new int[77];
    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;

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


  /**
   * Creates a new scanner
   *
   * @param   in  the java.io.Reader to read input from.
   */
  public AnalizadorLexico(java.io.Reader in) {
    this.zzReader = 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[0x110000];
  int i = 0;  /* index in packed string  */
  int j = 0;  /* index in unpacked array */
  while (i < 388) {
    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) {
    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) {
    /* if not: blow it up */
    char newBuffer[] = new char[zzBuffer.length*2];
    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) {
      throw new java.io.IOException("Reader returned 0 characters. See
JFlex examples for workaround.");
    }
    if (numRead > 0) {
      zzEndRead += numRead;
      /* If numRead == requested, we might have requested to few chars
to
         encode a full Unicode character. We assume that a Reader
would
         otherwise never return half characters. */
      if (numRead == requested) {
        if (Character.isHighSurrogate(zzBuffer[zzEndRead - 1])) {
          --zzEndRead;
          zzFinalHighSurrogate = 1;
        }
      }
      /* potentially more input available */
      return false;
    }

    /* numRead < 0 ==> end of stream */
    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>.
   *
   * 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;
    zzAtBOL  = true;
    zzAtEOF  = false;
```

```
    zzEOFDone = false;
    zzEndRead = zzStartRead = 0;
    zzCurrentPos = zzMarkedPos = 0;
    zzFinalHighSurrogate = 0;
    yyline = yychar = yycolumn = 0;
    zzLexicalState = YYINITIAL;
    if (zzBuffer.length > ZZ_BUFFERSIZE)
      zzBuffer = new char[ZZ_BUFFERSIZE];
  }


  /**
   * 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;
  }


  /**
   * 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
   */
  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;
    char [] zzCMapL = ZZ_CMAP;

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

    while (true) {
      zzMarkedPosL = zzMarkedPos;

      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] + zzCMapL[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:
            { System.out.print(yytext() + "[Caracter Invalido]");
            }
          case 23: break;
          case 2:
            { return new Symbol(sym.DIR, yytext());
            }
          case 24: break;
          case 3:
            { return new Symbol(sym.NUMBER,
Integer.valueOf(yytext()));
            }
          case 25: break;
          case 4:
```

```
    { return new Symbol(sym.LPAREN);
    }
  case 26: break;
  case 5:
    { return new Symbol(sym.RPAREN);
    }
  case 27: break;
  case 6:
    { return new Symbol(sym.COMMA);
    }
  case 28: break;
  case 7:
    { return new Symbol(sym.SCOLON);
    }
  case 29: break;
  case 8:
    { return new Symbol(sym.LBRACE);
    }
  case 30: break;
  case 9:
    { return new Symbol(sym.RBRACE);
    }
  case 31: break;
  case 10:
    { return new Symbol(sym.NEQ);
    }
  case 32: break;
  case 11:
    {
    }
  case 33: break;
  case 12:
    { return new Symbol(sym.IF);
    }
  case 34: break;
  case 13:
    { return new Symbol(sym.END);
    }
  case 35: break;
  case 14:
    { return new Symbol(sym.MOVE);
    }
  case 36: break;
  case 15:
    { return new Symbol(sym.START);
    }
  case 37: break;
  case 16:
    { return new Symbol(sym.WHILE);
    }
  case 38: break;
  case 17:
    { return new Symbol(sym.ITERATE);
    }
  case 39: break;
  case 18:
    { return new Symbol(sym.TURNLEFT);
```

```
            }
          case 40: break;
          case 19:
            { return new Symbol(sym.DRAWGRID);
            }
          case 41: break;
          case 20:
            { return new Symbol(sym.PLACEWALL);
            }
          case 42: break;
          case 21:
            { return new Symbol(sym.FCLEAR);
            }
          case 43: break;
          case 22:
            { return new Symbol(sym.PLACEROBOT);
            }
          case 44: break;
          default:
            zzScanError(ZZ_NO_MATCH);
        }
      }
    }
  }

  /**
   * Runs the scanner on input files.
   *
   * This is a standalone scanner, it will print any unmatched
   * text to System.out unchanged.
   *
   * @param argv   the command line, contains the filenames to run
   *               the scanner on.
   */
  public static void main(String argv[]) {
    if (argv.length == 0) {
      System.out.println("Usage : java AnalizadorLexico [ --encoding
<name> ] <inputfile(s)>");
    }
    else {
      int firstFilePos = 0;
      String encodingName = "UTF-8";
      if (argv[0].equals("--encoding")) {
        firstFilePos = 2;
        encodingName = argv[1];
        try {
          java.nio.charset.Charset.forName(encodingName); // Side-
effect: is encodingName valid?
        } catch (Exception e) {
          System.out.println("Invalid encoding '" + encodingName +
"'");
          return;
        }
      }
      for (int i = firstFilePos; i < argv.length; i++) {
        AnalizadorLexico scanner = null;
        try {
```

```
          java.io.FileInputStream stream = new
java.io.FileInputStream(argv[i]);
          java.io.Reader reader = new
java.io.InputStreamReader(stream, encodingName);
          scanner = new AnalizadorLexico(reader);
          while ( !scanner.zzAtEOF ) scanner.next_token();
        }
        catch (java.io.FileNotFoundException e) {
          System.out.println("File not found : \""+argv[i]+"\"");
        }
        catch (java.io.IOException e) {
          System.out.println("IO error scanning file
\""+argv[i]+"\"");
          System.out.println(e);
        }
        catch (Exception e) {
          System.out.println("Unexpected exception:");
          e.printStackTrace();
        }
      }
    }
  }


}
```

AnalizadorSintactico.java

```
//----------------------------------------------------
// The following code was generated by CUP v0.11a beta 20060608
// Thu Dec 03 02:58:16 JST 2015
//----------------------------------------------------

package compilador.modelos;

import java_cup.runtime.*;

/** CUP v0.11a beta 20060608 generated parser.
  * @version Thu Dec 03 02:58:16 JST 2015
  */
public class AnalizadorSintactico extends java_cup.runtime.lr_parser {

  /** Default constructor. */
  public AnalizadorSintactico() {super();}

  /** Constructor which sets the default scanner. */
  public AnalizadorSintactico(java_cup.runtime.Scanner s) {super(s);}

  /** Constructor which sets the default scanner. */
  public AnalizadorSintactico(java_cup.runtime.Scanner s,
java_cup.runtime.SymbolFactory sf) {super(s,sf);}

  /** Production table. */
  protected static final short _production_table[][] =
    unpackFromStrings(new String[] {
    "\000\025\000\002\002\006\000\002\002\004\000\002\003" +
```

```
    "\005\000\002\016\011\000\002\005\004\000\002\005\002" +
    "\000\002\006\011\000\002\004\013\000\002\007\004\000" +
    "\002\007\003\000\002\010\003\000\002\010\003\000\002" +
    "\010\003\000\002\010\003\000\002\011\011\000\002\013" +
    "\011\000\002\012\011\000\002\015\003\000\002\015\004" +
    "\000\002\014\004\000\002\014\004" });

  /** Access to production table. */
  public short[][] production_table() {return _production_table;}

  /** Parse-action table. */
  protected static final short[][] _action_table =
    unpackFromStrings(new String[] {
    "\000\104\000\004\011\004\001\002\000\004\017\101\001" +
    "\002\000\004\002\100\001\002\000\004\007\033\001\002" +
    "\000\006\012\ufffc\013\ufffc\001\002\000\006\012\014\013" +
    "\012\001\002\000\006\012\ufffd\013\ufffd\001\002\000\004" +
    "\017\025\001\002\000\004\007\ufffe\001\002\000\004\017" +
    "\015\001\002\000\004\027\016\001\002\000\004\021\017" +
    "\001\002\000\004\027\020\001\002\000\004\021\021\001" +
    "\002\000\004\026\022\001\002\000\004\020\023\001\002" +
    "\000\004\022\024\001\002\000\004\007\ufff9\001\002\000" +
    "\004\027\026\001\002\000\004\021\027\001\002\000\004" +
    "\027\030\001\002\000\004\020\031\001\002\000\004\022" +
    "\032\001\002\000\006\012\ufffb\013\ufffb\001\002\000\014" +
    "\004\043\005\035\006\046\014\036\015\044\001\002\000" +
    "\016\004\ufff4\005\ufff4\006\ufff4\010\ufff4\014\ufff4\015\ufff4"
+
    "\001\002\000\004\017\072\001\002\000\004\022\071\001" +
    "\002\000\016\004\ufff7\005\ufff7\006\ufff7\010\ufff7\014\ufff7" +
    "\015\ufff7\001\002\000\016\004\ufff5\005\ufff5\006\ufff5\010" +
    "\ufff5\014\ufff5\015\ufff5\001\002\000\016\004\043\005\035" +
    "\006\046\010\070\014\036\015\044\001\002\000\016\004" +
    "\ufff6\005\ufff6\006\ufff6\010\ufff6\014\ufff6\015\ufff6\001\002"
+
    "\000\004\017\056\001\002\000\004\022\055\001\002\000" +
    "\016\004\ufff8\005\ufff8\006\ufff8\010\ufff8\014\ufff8\015\ufff8"
+
    "\001\002\000\004\017\047\001\002\000\004\027\050\001" +
    "\002\000\004\020\051\001\002\000\004\024\052\001\002" +
    "\000\006\014\036\015\044\001\002\000\004\023\054\001" +
    "\002\000\016\004\ufff3\005\ufff3\006\ufff3\010\ufff3\014\ufff3" +
    "\015\ufff3\001\002\000\020\004\uffed\005\uffed\006\uffed\010" +
    "\uffed\014\uffed\015\uffed\023\uffed\001\002\000\006\016\060" +
    "\025\061\001\002\000\004\020\063\001\002\000\004\020" +
    "\ufff0\001\002\000\004\016\062\001\002\000\004\020\uffef" +
    "\001\002\000\004\024\064\001\002\000\006\014\036\015" +
    "\044\001\002\000\004\023\066\001\002\000\016\004\ufff2" +
    "\005\ufff2\006\ufff2\010\ufff2\014\ufff2\015\ufff2\001\002\000" +
    "\016\004\ufff9\005\ufff9\006\ufff9\010\ufff9\014\ufff9\015\ufff9"
+
    "\001\002\000\004\002\001\001\002\000\020\004\uffee\005" +
    "\uffee\006\uffee\010\uffee\014\uffee\015\uffee\023\uffee\001\002"
+
    "\000\006\016\060\025\061\001\002\000\004\020\074\001" +
    "\002\000\004\024\075\001\002\000\006\014\036\015\044" +
    "\001\002\000\004\023\077\001\002\000\016\004\ufff1\005" +
```

```
    "\ufff1\006\ufff1\010\ufff1\014\ufff1\015\ufff1\001\002\000\004" +
    "\002\000\001\002\000\004\027\102\001\002\000\004\021" +
    "\103\001\002\000\004\027\104\001\002\000\004\020\105" +
    "\001\002\000\004\022\106\001\002\000\006\012\ufffe\013" +
    "\ufffe\001\002" });

  /** Access to parse-action table. */
  public short[][] action_table() {return _action_table;}

  /** <code>reduce_goto</code> table. */
  protected static final short[][] _reduce_table =
    unpackFromStrings(new String[] {
    "\000\104\000\010\002\004\003\005\016\006\001\001\000" +
    "\002\001\001\000\002\001\001\000\002\001\001\000\004" +
    "\005\007\001\001\000\006\004\012\006\010\001\001\000" +
    "\002\001\001\000\002\001\001\000\002\001\001\000\002" +
    "\001\001\000\002\001\001\000\002\001\001\000\002\001" +
    "\001\000\002\001\001\000\002\001\001\000\002\001\001" +
    "\000\002\001\001\000\002\001\001\000\002\001\001\000" +
    "\002\001\001\000\002\001\001\000\002\001\001\000\002" +
    "\001\001\000\002\001\001\000\016\007\040\010\044\011" +
    "\036\012\041\013\037\014\033\001\001\000\002\001\001" +
    "\000\002\001\001\000\002\001\001\000\002\001\001\000" +
    "\002\001\001\000\014\010\066\011\036\012\041\013\037" +
    "\014\033\001\001\000\002\001\001\000\002\001\001\000" +
    "\002\001\001\000\002\001\001\000\002\001\001\000\002" +
    "\001\001\000\002\001\001\000\002\001\001\000\004\014" +
    "\052\001\001\000\002\001\001\000\002\001\001\000\002" +
    "\001\001\000\004\015\056\001\001\000\002\001\001\000" +
    "\002\001\001\000\002\001\001\000\002\001\001\000\002" +
    "\001\001\000\004\014\064\001\001\000\002\001\001\000" +
    "\002\001\001\000\002\001\001\000\002\001\001\000\002" +
    "\001\001\000\004\015\072\001\001\000\002\001\001\000" +
    "\002\001\001\000\004\014\075\001\001\000\002\001\001" +
    "\000\002\001\001\000\002\001\001\000\002\001\001\000" +
    "\002\001\001\000\002\001\001\000\002\001\001\000\002" +
    "\001\001\000\002\001\001" });

  /** Access to <code>reduce_goto</code> table. */
  public short[][] reduce_table() {return _reduce_table;}

  /** Instance of action encapsulation class. */
  protected CUP$AnalizadorSintactico$actions action_obj;

  /** Action encapsulation object initializer. */
  protected void init_actions()
    {
      action_obj = new CUP$AnalizadorSintactico$actions(this);
    }

  /** Invoke a user supplied parse action. */
  public java_cup.runtime.Symbol do_action(
    int                        act_num,
    java_cup.runtime.lr_parser parser,
    java.util.Stack            stack,
    int                        top)
    throws java.lang.Exception
```

```
  {
    /* call code in generated class */
    return action_obj.CUP$AnalizadorSintactico$do_action(act_num,
parser, stack, top);
  }

  /** Indicates start state. */
  public int start_state() {return 0;}
  /** Indicates start production. */
  public int start_production() {return 1;}

  /** <code>EOF</code> Symbol index. */
  public int EOF_sym() {return 0;}

  /** <code>error</code> Symbol index. */
  public int error_sym() {return 1;}

}

/** Cup generated class to encapsulate user supplied action code.*/
class CUP$AnalizadorSintactico$actions {



      public void forFunc (int num, int instr) {
            for (int i = 0; i < num; i++) {
                  if (instr==0) GridMap.move();
                  else if (instr==1) GridMap.turnLeft();
                  else throw new IllegalArgumentException("forFunc:
Illegal Instruction");
                  }
      }

/*    public void loopFunc (int inst, int cond, int func) {
            if (func == 0 {

       if (cond == 0) {
                                    while (GridMap.isFrontClear()) {
if (inst == 0) GridMap.move(); else GridMap.turnLeft(); }
                                    }
                        else { while (!GridMap.isFrontClear()) { if
(inst == 0) GridMap.move(); else GridMap.turnLeft(); }
                        }
      }
*/


  private final AnalizadorSintactico parser;

  /** Constructor */
  CUP$AnalizadorSintactico$actions(AnalizadorSintactico parser) {
    this.parser = parser;
  }

  /** Method with the actual generated action code. */
  public final java_cup.runtime.Symbol
CUP$AnalizadorSintactico$do_action(
```

```
    int                        CUP$AnalizadorSintactico$act_num,
    java_cup.runtime.lr_parser CUP$AnalizadorSintactico$parser,
    java.util.Stack            CUP$AnalizadorSintactico$stack,
    int                        CUP$AnalizadorSintactico$top)
    throws java.lang.Exception
    {
      /* Symbol object for return from actions */
      java_cup.runtime.Symbol CUP$AnalizadorSintactico$result;

      /* select the action based on the action number */
      switch (CUP$AnalizadorSintactico$act_num)
        {
          /*. . . . . .*/
          case 20: // instruction ::= TURNLEFT SCOLON
            {
              Integer RESULT =null;
             RESULT = 1;
              CUP$AnalizadorSintactico$result =
parser.getSymbolFactory().newSymbol("instruction",10,
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-1)),
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
RESULT);
            }
          return CUP$AnalizadorSintactico$result;

          /*. . . . . .*/
          case 19: // instruction ::= MOVE SCOLON
            {
              Integer RESULT =null;
             RESULT = 0;
              CUP$AnalizadorSintactico$result =
parser.getSymbolFactory().newSymbol("instruction",10,
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-1)),
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
RESULT);
            }
          return CUP$AnalizadorSintactico$result;

          /*. . . . . .*/
          case 18: // cond ::= NEQ FCLEAR
            {
              Integer RESULT =null;
             RESULT = 1;
              CUP$AnalizadorSintactico$result =
parser.getSymbolFactory().newSymbol("cond",11,
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-1)),
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
RESULT);
            }
          return CUP$AnalizadorSintactico$result;

          /*. . . . . .*/
          case 17: // cond ::= FCLEAR
            {
```

```
              Integer RESULT =null;
             RESULT = 0;
              CUP$AnalizadorSintactico$result =
parser.getSymbolFactory().newSymbol("cond",11,
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
RESULT);
            }
          return CUP$AnalizadorSintactico$result;

          /*. . . . . .*/
          case 16: // loop ::= WHILE LPAREN cond RPAREN LBRACE
instruction RBRACE
            {
              Object RESULT =null;
            int bleft =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-4)).left;
            int bright =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-4)).right;
            Integer b = (Integer)((java_cup.runtime.Symbol)
CUP$AnalizadorSintactico$stack.elementAt(CUP$AnalizadorSintactico$top-
4)).value;
            int ileft =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-1)).left;
            int iright =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-1)).right;
            Integer i = (Integer)((java_cup.runtime.Symbol)
CUP$AnalizadorSintactico$stack.elementAt(CUP$AnalizadorSintactico$top-
1)).value;
                  if (b == 0) {
                                    while (GridMap.isFrontClear()) {
if (i == 0) GridMap.move(); else GridMap.turnLeft(); }
                                    }
                        else { while (!GridMap.isFrontClear()) { if (i
== 0) GridMap.move(); else GridMap.turnLeft(); }
                        }

              CUP$AnalizadorSintactico$result =
parser.getSymbolFactory().newSymbol("loop",8,
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-6)),
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
RESULT);
            }
          return CUP$AnalizadorSintactico$result;

          /*. . . . . .*/
          case 15: // conditional ::= IF LPAREN cond RPAREN LBRACE
instruction RBRACE
            {
              Object RESULT =null;
```

```
            int bleft =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-4)).left;
            int bright =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-4)).right;
            Integer b = (Integer)((java_cup.runtime.Symbol)
CUP$AnalizadorSintactico$stack.elementAt(CUP$AnalizadorSintactico$top-
4)).value;
            int ileft =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-1)).left;
            int iright =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-1)).right;
            Integer i = (Integer)((java_cup.runtime.Symbol)
CUP$AnalizadorSintactico$stack.elementAt(CUP$AnalizadorSintactico$top-
1)).value;
                    if (b == 0) {
                                        if (GridMap.isFrontClear()) { if
(i == 0) GridMap.move(); else GridMap.turnLeft(); }
                                        }
                            else { if (!GridMap.isFrontClear()) { if (i ==
0) GridMap.move(); else GridMap.turnLeft(); }
                            }

              CUP$AnalizadorSintactico$result =
parser.getSymbolFactory().newSymbol("conditional",9,
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-6)),
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
RESULT);
            }
          return CUP$AnalizadorSintactico$result;

          /*. . . . . .*/
          case 14: // iteration ::= ITERATE LPAREN NUMBER RPAREN
LBRACE instruction RBRACE
            {
              Object RESULT =null;
            int nleft =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-4)).left;
            int nright =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-4)).right;
            Integer n = (Integer)((java_cup.runtime.Symbol)
CUP$AnalizadorSintactico$stack.elementAt(CUP$AnalizadorSintactico$top-
4)).value;
            int ileft =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-1)).left;
            int iright =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-1)).right;
```

```
            Integer i = (Integer)((java_cup.runtime.Symbol)
CUP$AnalizadorSintactico$stack.elementAt(CUP$AnalizadorSintactico$top-
1)).value;
              forFunc(n, i);
              CUP$AnalizadorSintactico$result =
parser.getSymbolFactory().newSymbol("iteration",7,
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-6)),
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
RESULT);
            }
          return CUP$AnalizadorSintactico$result;

          /*. . . . . .*/
          case 13: // statement ::= instruction
            {
              Object RESULT =null;
            int ileft =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()).left;
            int iright =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()).right
;
            Integer i = (Integer)((java_cup.runtime.Symbol)
CUP$AnalizadorSintactico$stack.peek()).value;
             if (i == 0) GridMap.move(); else GridMap.turnLeft();
              CUP$AnalizadorSintactico$result =
parser.getSymbolFactory().newSymbol("statement",6,
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
RESULT);
            }
          return CUP$AnalizadorSintactico$result;

          /*. . . . . .*/
          case 12: // statement ::= conditional
            {
              Object RESULT =null;

              CUP$AnalizadorSintactico$result =
parser.getSymbolFactory().newSymbol("statement",6,
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
RESULT);
            }
          return CUP$AnalizadorSintactico$result;

          /*. . . . . .*/
          case 11: // statement ::= loop
            {
              Object RESULT =null;

              CUP$AnalizadorSintactico$result =
parser.getSymbolFactory().newSymbol("statement",6,
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
RESULT);
            }
```

```
            return CUP$AnalizadorSintactico$result;

            /*. . . . . . . . . . . . . . . . . . . .*/
            case 10: // statement ::= iteration
              {
                Object RESULT =null;

                CUP$AnalizadorSintactico$result =
parser.getSymbolFactory().newSymbol("statement",6,
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
RESULT);
              }
            return CUP$AnalizadorSintactico$result;

            /*. . . . . . . . . . . . . . . . . . . .*/
            case 9: // statementList ::= statement
              {
                Object RESULT =null;

                CUP$AnalizadorSintactico$result =
parser.getSymbolFactory().newSymbol("statementList",5,
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
RESULT);
              }
            return CUP$AnalizadorSintactico$result;

            /*. . . . . . . . . . . . . . . . . . . .*/
            case 8: // statementList ::= statementList statement
              {
                Object RESULT =null;

                CUP$AnalizadorSintactico$result =
parser.getSymbolFactory().newSymbol("statementList",5,
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-1)),
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
RESULT);
              }
            return CUP$AnalizadorSintactico$result;

            /*. . . . . . . . . . . . . . . . . . . .*/
            case 7: // robot ::= PLACEROBOT LPAREN NUMBER COMMA NUMBER
COMMA DIR RPAREN SCOLON
              {
                Object RESULT =null;
              int n1left =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-6)).left;
              int n1right =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-6)).right;
              Integer n1 = (Integer)((java_cup.runtime.Symbol)
CUP$AnalizadorSintactico$stack.elementAt(CUP$AnalizadorSintactico$top-
6)).value;
```

```
            int n2left =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-4)).left;
            int n2right =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-4)).right;
            Integer n2 = (Integer)((java_cup.runtime.Symbol)
CUP$AnalizadorSintactico$stack.elementAt(CUP$AnalizadorSintactico$top-
4)).value;
            int dleft =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-2)).left;
            int dright =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-2)).right;
            String d = (String)((java_cup.runtime.Symbol)
CUP$AnalizadorSintactico$stack.elementAt(CUP$AnalizadorSintactico$top-
2)).value;
             GridMap.setElement(n1,n2,d.charAt(0));
              CUP$AnalizadorSintactico$result =
parser.getSymbolFactory().newSymbol("robot",2,
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-8)),
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
RESULT);
            }
          return CUP$AnalizadorSintactico$result;

          /*. . . . . .*/
          case 6: // wall ::= PLACEWALL LPAREN NUMBER COMMA NUMBER
RPAREN SCOLON
            {
              Object RESULT =null;
            int n1left =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-4)).left;
            int n1right =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-4)).right;
            Integer n1 = (Integer)((java_cup.runtime.Symbol)
CUP$AnalizadorSintactico$stack.elementAt(CUP$AnalizadorSintactico$top-
4)).value;
            int n2left =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-2)).left;
            int n2right =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-2)).right;
            Integer n2 = (Integer)((java_cup.runtime.Symbol)
CUP$AnalizadorSintactico$stack.elementAt(CUP$AnalizadorSintactico$top-
2)).value;
              GridMap.setElement(n1,n2,"Wall");
              CUP$AnalizadorSintactico$result =
parser.getSymbolFactory().newSymbol("wall",4,
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-6)),
```

```
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
RESULT);
            }
          return CUP$AnalizadorSintactico$result;

          /*. . . . . .*/
          case 5: // wallList ::=
            {
              Object RESULT =null;

              CUP$AnalizadorSintactico$result =
parser.getSymbolFactory().newSymbol("wallList",3,
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
RESULT);
            }
          return CUP$AnalizadorSintactico$result;

          /*. . . . . .*/
          case 4: // wallList ::= wallList wall
            {
              Object RESULT =null;

              CUP$AnalizadorSintactico$result =
parser.getSymbolFactory().newSymbol("wallList",3,
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-1)),
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
RESULT);
            }
          return CUP$AnalizadorSintactico$result;

          /*. . . . . .*/
          case 3: // grid ::= DRAWGRID LPAREN NUMBER COMMA NUMBER
RPAREN SCOLON
            {
              GridMap RESULT =null;
            int n1left =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-4)).left;
            int n1right =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-4)).right;
            Integer n1 = (Integer)((java_cup.runtime.Symbol)
CUP$AnalizadorSintactico$stack.elementAt(CUP$AnalizadorSintactico$top-
4)).value;
            int n2left =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-2)).left;
            int n2right =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-2)).right;
            Integer n2 = (Integer)((java_cup.runtime.Symbol)
CUP$AnalizadorSintactico$stack.elementAt(CUP$AnalizadorSintactico$top-
2)).value;
             RESULT = new GridMap(n1,n2);
```

```
              CUP$AnalizadorSintactico$result =
parser.getSymbolFactory().newSymbol("grid",12,
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-6)),
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
RESULT);
            }
          return CUP$AnalizadorSintactico$result;

          /*. . . . . .*/
          case 2: // initial ::= grid wallList robot
            {
              Object RESULT =null;
             GridMap.printState();
              CUP$AnalizadorSintactico$result =
parser.getSymbolFactory().newSymbol("initial",1,
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-2)),
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
RESULT);
            }
          return CUP$AnalizadorSintactico$result;

          /*. . . . . .*/
          case 1: // $START ::= program EOF
            {
              Object RESULT =null;
            int start_valleft =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-1)).left;
            int start_valright =
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-1)).right;
            Object start_val = (Object)((java_cup.runtime.Symbol)
CUP$AnalizadorSintactico$stack.elementAt(CUP$AnalizadorSintactico$top-
1)).value;
            RESULT = start_val;
              CUP$AnalizadorSintactico$result =
parser.getSymbolFactory().newSymbol("$START",0,
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-1)),
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
RESULT);
            }
          /* ACCEPT */
          CUP$AnalizadorSintactico$parser.done_parsing();
          return CUP$AnalizadorSintactico$result;

          /*. . . . . .*/
          case 0: // program ::= initial START statementList END
            {
              Object RESULT =null;
             System.out.println("[Parse completed succesfully]");
              CUP$AnalizadorSintactico$result =
parser.getSymbolFactory().newSymbol("program",0,
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.elementAt(CUP
$AnalizadorSintactico$top-3)),
```

```
((java_cup.runtime.Symbol)CUP$AnalizadorSintactico$stack.peek()),
RESULT);
            }
          return CUP$AnalizadorSintactico$result;

          /* . . . . . .*/
          default:
            throw new Exception(
               "Invalid action number found in internal parse table");

        }
    }
}
```

Lotes de Prueba

Prueba 1:

Entrada:

```
drawGrid(5,5);
placeWall(0,1);
placeWall(2,3);
placeWall(9,9);
placeRobot(0,1,N);

START

iterate (3) { turnLeft; }
while (frontClear) {move;}
if (!frontClear) {turnLeft;}
if (!frontClear) {turnLeft;}
move; move; turnLeft;

END
```

Salida:

```
Position [9,9] Out of Bounds

[ ] [^] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [W] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
```

```
[ ] [<] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [W] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
```

```
[ ] [v] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [W] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
```

```
[ ] [>] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [W] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
```

```
[ ] [ ] [>] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [W] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
```

```
[ ] [ ] [ ] [>] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [W] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
```

```
[ ] [ ] [ ] [ ] [>]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [W] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
```

```
[ ] [ ] [ ] [ ] [^]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [W] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
```

```
[ ] [ ] [ ] [ ] [<]
```

```
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [W] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]


[ ] [ ] [ ] [<] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [W] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]


[ ] [ ] [<] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [W] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]


[ ] [ ] [v] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [W] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]

[Parse completed succesfully]
```

Prueba 2:

Error sintáctico, se provee un numero de dos dígitos al método drawGrid, por lo que el scanner lo separa en dos token (1 y 0). Esto desembocara en un error sintáctico cuando el parser trate de reducirlo a la regla, por lo que se origina el error de Cup.

```
drawGrid(10,10);
placeWall(0,1);
placeWall(2,3);
placeWall(9,9);
placeRobot(0,1,N);

START
iterate (3) { turnLeft; }
while (frontClear) {move;}
if (!frontClear) {turnLeft;}
if (!frontClear) {turnLeft;}
move; move; turnLeft;
END
```

Syntax error

```
Couldn't repair and continue parse
java.lang.Exception: Can't recover from previous error(s)
      at
java_cup.runtime.lr_parser.report_fatal_error(lr_parser.java:375)
      at
java_cup.runtime.lr_parser.unrecovered_syntax_error(lr_parser.java:424
)
      at java_cup.runtime.lr_parser.parse(lr_parser.java:616)
      at compilador.controladores.Main.main(Main.java:13)
```

Prueba 3:

Error lexico 'placRobot' no es reconocido por el Jflex como un lexema valido. Por lo tanto al tratar de reducir la regla se encontrara con un error de parsing, ya que el Jflex no le pasa el token correcto al Cup.

```
drawGrid(9,9);
placeWall(0,1);
placeWall(2,3);
placeWall(9,9);
placRobot(0,1,N);

START
iterate (3) { turnLeft; }
while (frontClear) { move; }
if (!frontClear) {turnLeft;}
if (!frontClear) {turnLeft;}
move; move; turnLeft;
END

p[Caracter Invalido]l[Caracter Invalido]a[Caracter Invalido]
Syntax error
Couldn't repair and continue parse
c[Caracter Invalido]R[Caracter Invalido]o[Caracter Invalido]b[Caracter
Invalido]o[Caracter Invalido]t[Caracter Invalido]java.lang.Exception:
Can't recover from previous error(s)
      at
java_cup.runtime.lr_parser.report_fatal_error(lr_parser.java:375)
      at
java_cup.runtime.lr_parser.unrecovered_syntax_error(lr_parser.java:424
)
      at java_cup.runtime.lr_parser.parse(lr_parser.java:616)
      at compilador.controladores.Main.main(Main.java:13)
```

CONCLUSIONES

Jflex (el scanner) se encarga de dividir la entradas en tokens
Cup (el parser) chequea si la entrada es sintacticamente correcta y ejecuta acciones semanticas cuando reduce una regla

CUP implemente un parser LALR (Look-Ahead Left-to-right parser), que es una version simplificada del LR parser, el cual parsea un texto de acuerdo con producciones basada en una gramatica libre de contexto.
Este mantiene un stack (pila) de estados y una tabla con la cual decide si shiftea el token al stack o si reduce la produccion, basado en el próximo token (lookahead)

A alto nivel, la diferencia entre el parseo LL y el parseo LR es qué los parseadores LL empiezan en el símbolo de entrada e intentan aplicar producciones para llegar a la cadena destino. Mientras qué los parseadores LR empiezan en la cadena destino e intentan llegar al símbolo de entrada.
Un parseador LL es un parseador de izquierda a derecha, es un derivador de izquierda. Consideramos los simbolos desde la izquierda a la derecha e intentamos construir una derivacion más a la izquierda. Esto se hace comenzando con el simbolo de entrada y expandiendo repetidamente el no terminal más a la izquierda hasta llegar a la cadena destino. Un parseador LR es de izquierda a derecha, un derivador de más a la derecha. Lo cual significa qué escaneamos desde la izquierda a la derecha e intentamos construir una derivacion más a la derecha. El parser agarra continuamente una subcadena de la entrada e intenta revertirla devuelta a un no terminal.

Durante un parseo LL, el parser está continuamente eligiendo entre dos acciones

1. Predict: Basado en el no terminal más a la izquierda y un numero de tokens más adelante, elegir qué produccion deberia aplicar para acercarse más a la cadena de entrada.
2. Match: Concidir el terminal más a la izquierda con el simbolo no consumido más a la izquierda de la entrada.

En cada paso, nos fijamos en el simbolo más a la izquierda de nuestra produccion. Si es un terminal, entonces lo hacemos coincidir, si no es un terminal, entonces precedimos qué va a ser eligiendo una de las reglas.

En un parser LR, existen dos acciones:

1. Shift: Agregar el proximo token de entrada al buffer para su consideracion.

2. Reduce: Reduce una coleccion de terminales y no terminales en este buffer devuelta a un no terminal realizando una reversion sobre la produccion.

Como ejemplo, un parseador LR (con un token proximo) puede parsear la siguiente cadena de está forma:

Los dos algoritmos de parseo mencionados (LL and LR) son conocidos por tener diferentes características. Los parseadores LL tienden a ser más faciles de escribir a mano, pero tienen peor performance qué los parsers LR. A su vez, aceptan un conjunto gramatical mucho más chico qué los parseadores LR. Los parseadores LR vienen de muchas formas (LR(0),SRL(1),LALR(1),LR(1),IELR(1),GLR(0),etc) y su performance es mucho mayor.
Tambien tienden a ser mucho más complejos y por lo general son generados por herramientas como yacc o bison. Los parsers LL tambien tienen muchas formas (incluyendo LL(*), el cual es usado por la herramienta ANTLR), por lo tanto en la practica LL(1) es el más utilizado.

BIBLIOGRAFIA

Sergio Galves Rojas, Miguel Angel Mora Mata, "Java a Tope: Traductores y Compiladores con Lex/Yacc, JFlex/Cup y JavaCC", Universidad de Malaga, 2005

"Karel programming language documentation"
http://mormegil.wz.cz/prog/karel/prog_doc.htm
Ultimo Acceso: Noviembre 2015

"JFlex User's Manual"
http://jflex.de/manual.html
Ultimo Acceso: Noviembre 2015

"CUP User's Manual"
http://www2.cs.tum.edu/projects/cup/manual.html
Ultimo Acceso: Noviembre 2015

"What is the difference between LL and LR parsing?"
http://stackoverflow.com/questions/5975741/what-is-the-difference-between-ll-and-lr-parsing