```
//----------------------------------------------------
// The following code was generated by CUP v0.11b 20160615 (GIT 4ac7450)
//----------------------------------------------------

package analizadores;

import java_cup.runtime.*;
import java.util.LinkedList;
import arbol.*;
import excepciones.Errores;
import java_cup.runtime.XMLElement;

/** CUP v0.11b 20160615 (GIT 4ac7450) generated parser.
  */
@SuppressWarnings({"rawtypes"})
public class Parser extends java_cup.runtime.lr_parser {

 public final Class getSymbolContainer() {
   return sym.class;
}

  /** Default constructor. */
  @Deprecated
  public Parser() {super();}

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

  /** Constructor which sets the default scanner. */
  public Parser(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\106\000\002\002\004\000\002\002\003\000\002\003" +
    "\004\000\002\003\003\000\002\004\003\000\002\004\003" +
    "\000\002\004\003\000\002\004\003\000\002\004\003\000" +
    "\002\004\003\000\002\004\003\000\002\004\003\000\002" +
    "\004\003\000\002\004\003\000\002\004\003\000\002\004" +
    "\003\000\002\004\004\000\002\005\011\000\002\005\007" +
    "\000\002\006\006\000\002\022\005\000\002\022\004\000" +
    "\002\022\004\000\002\007\007\000\002\020\005\000\002" +
    "\020\005\000\002\010\011\000\002\010\015\000\002\010" +
    "\013\000\002\011\011\000\002\012\013\000\002\013\014" +
    "\000\002\015\004\000\002\016\004\000\002\017\006\000" +
    "\002\014\011\000\002\021\007\000\002\021\006\000\002" +
    "\021\002\000\002\024\003\000\002\024\003\000\002\024" +
    "\003\000\002\024\003\000\002\024\003\000\002\023\006" +
    "\000\002\023\004\000\002\023\005\000\002\023\005\000" +
    "\002\023\005\000\002\023\005\000\002\023\005\000\002" +
    "\023\005\000\002\023\005\000\002\023\005\000\002\023" +
    "\005\000\002\023\005\000\002\023\005\000\002\023\005" +
    "\000\002\023\005\000\002\023\005\000\002\023\005\000" +
    "\002\023\004\000\002\023\005\000\002\023\003\000\002" +
    "\023\003\000\002\023\003\000\002\023\003\000\002\023" +
```

```
  "\003\000\002\023\003\000\002\023\003" });

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

/** Parse-action table. */
protected static final short[][] _action_table =
  unpackFromStrings(new String[] {
  "\000\237\000\032\003\031\004\035\012\015\014\014\017" +
  "\010\020\034\021\021\022\025\023\006\025\030\026\032" +
  "\076\016\001\002\000\042\002\ufff3\003\ufff3\004\ufff3\012" +
  "\ufff3\014\ufff3\015\ufff3\016\ufff3\017\ufff3\020\ufff3\021\ufff3" +
  "\022\ufff3\023\ufff3\025\ufff3\026\ufff3\040\ufff3\076\ufff3\001" +
  "\002\000\034\002\000\003\031\004\035\012\015\014\014" +
  "\017\010\020\034\021\021\022\025\023\006\025\030\026" +
  "\032\076\016\001\002\000\004\034\241\001\002\000\042" +
  "\002\ufff7\003\ufff7\004\ufff7\012\ufff7\014\ufff7\015\ufff7\016" +
  "\ufff7\017\ufff7\020\ufff7\021\ufff7\022\ufff7\023\ufff7\025\ufff7" +
  "\026\ufff7\040\ufff7\076\ufff7\001\002\000\004\035\233\001" +
  "\002\000\042\002\ufffa\003\ufffa\004\ufffa\012\ufffa\014\ufffa" +
  "\015\ufffa\016\ufffa\017\ufffa\020\ufffa\021\ufffa\022\ufffa\023" +
  "\ufffa\025\ufffa\026\ufffa\040\ufffa\076\ufffa\001\002\000\004" +
  "\002\232\001\002\000\042\002\ufffe\003\ufffe\004\ufffe\012" +
  "\ufffe\014\ufffe\015\ufffe\016\ufffe\017\ufffe\020\ufffe\021\ufffe" +
  "\022\ufffe\023\ufffe\025\ufffe\026\ufffe\040\ufffe\076\ufffe\001" +
  "\002\000\004\035\215\001\002\000\004\035\202\001\002" +
  "\000\010\046\152\055\177\056\176\001\002\000\042\002" +
  "\ufffc\003\ufffc\004\ufffc\012\ufffc\014\ufffc\015\ufffc\016\ufffc" +
  "\017\ufffc\020\ufffc\021\ufffc\022\ufffc\023\ufffc\025\ufffc\026" +
  "\ufffc\040\ufffc\076\ufffc\001\002\000\042\002\ufff4\003\ufff4" +
  "\004\ufff4\012\ufff4\014\ufff4\015\ufff4\016\ufff4\017\ufff4\020" +
  "\ufff4\021\ufff4\022\ufff4\023\ufff4\025\ufff4\026\ufff4\040\ufff4" +
  "\076\ufff4\001\002\000\004\037\166\001\002\000\042\002" +
  "\ufffd\003\ufffd\004\ufffd\012\ufffd\014\ufffd\015\ufffd\016\ufffd" +
  "\017\ufffd\020\ufffd\021\ufffd\022\ufffd\023\ufffd\025\ufffd\026" +
  "\ufffd\040\ufffd\076\ufffd\001\002\000\042\002\ufff6\003\ufff6" +
  "\004\ufff6\012\ufff6\014\ufff6\015\ufff6\016\ufff6\017\ufff6\020" +
  "\ufff6\021\ufff6\022\ufff6\023\ufff6\025\ufff6\026\ufff6\040\ufff6" +
  "\076\ufff6\001\002\000\042\002\ufff2\003\ufff2\004\ufff2\012" +
  "\ufff2\014\ufff2\015\ufff2\016\ufff2\017\ufff2\020\ufff2\021\ufff2" +
  "\022\ufff2\023\ufff2\025\ufff2\026\ufff2\040\ufff2\076\ufff2\001" +
  "\002\000\004\034\165\001\002\000\042\002\ufff5\003\ufff5" +
  "\004\ufff5\012\ufff5\014\ufff5\015\ufff5\016\ufff5\017\ufff5\020" +
  "\ufff5\021\ufff5\022\ufff5\023\ufff5\025\ufff5\026\ufff5\040\ufff5" +
  "\076\ufff5\001\002\000\042\002\ufffb\003\ufffb\004\ufffb\012" +
  "\ufffb\014\ufffb\015\ufffb\016\ufffb\017\ufffb\020\ufffb\021\ufffb" +
  "\022\ufffb\023\ufffb\025\ufffb\026\ufffb\040\ufffb\076\ufffb\001" +
  "\002\000\004\035\161\001\002\000\004\034\160\001\002" +
  "\000\004\035\155\001\002\000\042\002\ufff8\003\ufff8\004" +
  "\ufff8\012\ufff8\014\ufff8\015\ufff8\016\ufff8\017\ufff8\020\ufff8" +
  "\021\ufff8\022\ufff8\023\ufff8\025\ufff8\026\ufff8\040\ufff8\076" +
  "\ufff8\001\002\000\004\035\132\001\002\000\004\076\037" +
  "\001\002\000\042\002\ufff9\003\ufff9\004\ufff9\012\ufff9\014" +
  "\ufff9\015\ufff9\016\ufff9\017\ufff9\020\ufff9\021\ufff9\022\ufff9" +
  "\023\ufff9\025\ufff9\026\ufff9\040\ufff9\076\ufff9\001\002\000" +
  "\004\043\040\001\002\000\014\005\046\006\041\007\044" +
  "\010\045\011\042\001\002\000\010\034\uffd9\036\uffd9\046" +
```

"\uffd9\001\002\000\010\034\uffd6\036\uffd6\046\uffd6\001\002" +
"\000\006\034\050\046\047\001\002\000\010\034\uffd8\036" +
"\uffd8\046\uffd8\001\002\000\010\034\uffd7\036\uffd7\046\uffd7" +
"\001\002\000\010\034\uffda\036\uffda\046\uffda\001\002\000" +
"\026\027\055\030\060\035\051\050\053\067\063\072\054" +
"\073\052\074\062\075\056\076\061\001\002\000\042\002" +
"\uffef\003\uffef\004\uffef\012\uffef\014\uffef\015\uffef\016\uffef" +
"\017\uffef\020\uffef\021\uffef\022\uffef\023\uffef\025\uffef\026" +
"\uffef\040\uffef\076\uffef\001\002\000\040\005\046\006\041" +
"\007\044\010\045\011\042\027\055\030\060\035\051\050" +
"\053\067\063\072\054\073\052\074\062\075\056\076\061" +
"\001\002\000\046\034\uffc1\036\uffc1\043\uffc1\047\uffc1\050" +
"\uffc1\051\uffc1\052\uffc1\053\uffc1\054\uffc1\057\uffc1\060\uffc1" +
"\061\uffc1\062\uffc1\063\uffc1\064\uffc1\065\uffc1\066\uffc1\070" +
"\uffc1\001\002\000\026\027\055\030\060\035\051\050\053" +
"\067\063\072\054\073\052\074\062\075\056\076\061\001" +
"\002\000\046\034\uffc2\036\uffc2\043\uffc2\047\uffc2\050\uffc2" +
"\051\uffc2\052\uffc2\053\uffc2\054\uffc2\057\uffc2\060\uffc2\061" +
"\uffc2\062\uffc2\063\uffc2\064\uffc2\065\uffc2\066\uffc2\070\uffc2" +
"\001\002\000\046\034\uffbe\036\uffbe\043\uffbe\047\uffbe\050" +
"\uffbe\051\uffbe\052\uffbe\053\uffbe\054\uffbe\057\uffbe\060\uffbe" +
"\061\uffbe\062\uffbe\063\uffbe\064\uffbe\065\uffbe\066\uffbe\070" +
"\uffbe\001\002\000\046\034\uffbf\036\uffbf\043\uffbf\047\uffbf" +
"\050\uffbf\051\uffbf\052\uffbf\053\uffbf\054\uffbf\057\uffbf\060" +
"\uffbf\061\uffbf\062\uffbf\063\uffbf\064\uffbf\065\uffbf\066\uffbf" +
"\070\uffbf\001\002\000\042\034\123\047\075\050\070\051" +
"\074\052\077\053\073\054\071\057\101\060\102\061\072" +
"\062\065\063\100\064\076\065\066\066\103\070\067\001" +
"\002\000\046\034\uffbd\036\uffbd\043\uffbd\047\uffbd\050\uffbd" +
"\051\uffbd\052\uffbd\053\uffbd\054\uffbd\057\uffbd\060\uffbd\061" +
"\uffbd\062\uffbd\063\uffbd\064\uffbd\065\uffbd\066\uffbd\070\uffbd" +
"\001\002\000\046\034\uffbc\036\uffbc\043\uffbc\047\uffbc\050" +
"\uffbc\051\uffbc\052\uffbc\053\uffbc\054\uffbc\057\uffbc\060\uffbc" +
"\061\uffbc\062\uffbc\063\uffbc\064\uffbc\065\uffbc\066\uffbc\070" +
"\uffbc\001\002\000\046\034\uffc0\036\uffc0\043\uffc0\047\uffc0" +
"\050\uffc0\051\uffc0\052\uffc0\053\uffc0\054\uffc0\057\uffc0\060" +
"\uffc0\061\uffc0\062\uffc0\063\uffc0\064\uffc0\065\uffc0\066\uffc0" +
"\070\uffc0\001\002\000\026\027\055\030\060\035\051\050" +
"\053\067\063\072\054\073\052\074\062\075\056\076\061" +
"\001\002\000\046\034\uffc4\036\uffc4\043\uffc4\047\075\050" +
"\070\051\074\052\077\053\073\054\071\057\101\060\102" +
"\061\072\062\065\063\100\064\076\065\uffc4\066\uffc4\070" +
"\uffc4\001\002\000\026\027\055\030\060\035\051\050\053" +
"\067\063\072\054\073\052\074\062\075\056\076\061\001" +
"\002\000\026\027\055\030\060\035\051\050\053\067\063" +
"\072\054\073\052\074\062\075\056\076\061\001\002\000" +
"\026\027\055\030\060\035\051\050\053\067\063\072\054" +
"\073\052\074\062\075\056\076\061\001\002\000\026\027" +
"\055\030\060\035\051\050\053\067\063\072\054\073\052" +
"\074\062\075\056\076\061\001\002\000\026\027\055\030" +
"\060\035\051\050\053\067\063\072\054\073\052\074\062" +
"\075\056\076\061\001\002\000\026\027\055\030\060\035" +
"\051\050\053\067\063\072\054\073\052\074\062\075\056" +
"\076\061\001\002\000\026\027\055\030\060\035\051\050" +
"\053\067\063\072\054\073\052\074\062\075\056\076\061" +
"\001\002\000\026\027\055\030\060\035\051\050\053\067" +
"\063\072\054\073\052\074\062\075\056\076\061\001\002" +

"\000\026\027\055\030\060\035\051\050\053\067\063\072" +
"\054\073\052\074\062\075\056\076\061\001\002\000\026" +
"\027\055\030\060\035\051\050\053\067\063\072\054\073" +
"\052\074\062\075\056\076\061\001\002\000\026\027\055" +
"\030\060\035\051\050\053\067\063\072\054\073\052\074" +
"\062\075\056\076\061\001\002\000\026\027\055\030\060" +
"\035\051\050\053\067\063\072\054\073\052\074\062\075" +
"\056\076\061\001\002\000\026\027\055\030\060\035\051" +
"\050\053\067\063\072\054\073\052\074\062\075\056\076" +
"\061\001\002\000\026\027\055\030\060\035\051\050\053" +
"\067\063\072\054\073\052\074\062\075\056\076\061\001" +
"\002\000\026\027\055\030\060\035\051\050\053\067\063" +
"\072\054\073\052\074\062\075\056\076\061\001\002\000" +
"\046\034\uffc6\036\uffc6\043\uffc6\047\075\050\070\051\074" +
"\052\077\053\073\054\071\057\101\060\102\061\072\062" +
"\065\063\100\064\076\065\066\066\uffc6\070\067\001\002" +
"\000\046\034\uffc8\036\uffc8\043\uffc8\047\075\050\070\051" +
"\074\052\077\053\073\054\071\057\uffc8\060\uffc8\061\uffc8" +
"\062\uffc8\063\uffc8\064\uffc8\065\uffc8\066\uffc8\070\uffc8\001" +
"\002\000\046\034\uffc9\036\uffc9\043\uffc9\047\075\050\070" +
"\051\074\052\077\053\073\054\071\057\uffc9\060\uffc9\061" +
"\uffc9\062\uffc9\063\uffc9\064\uffc9\065\uffc9\066\uffc9\070\uffc9" +
"\001\002\000\046\034\uffcd\036\uffcd\043\uffcd\047\075\050" +
"\070\051\074\052\077\053\073\054\071\057\uffcd\060\uffcd" +
"\061\uffcd\062\uffcd\063\uffcd\064\uffcd\065\uffcd\066\uffcd\070" +
"\uffcd\001\002\000\046\034\uffd0\036\uffd0\043\uffd0\047\uffd0" +
"\050\uffd0\051\uffd0\052\uffd0\053\uffd0\054\071\057\uffd0\060" +
"\uffd0\061\uffd0\062\uffd0\063\uffd0\064\uffd0\065\uffd0\066\uffd0" +
"\070\uffd0\001\002\000\046\034\uffcb\036\uffcb\043\uffcb\047" +
"\075\050\070\051\074\052\077\053\073\054\071\057\uffcb" +
"\060\uffcb\061\uffcb\062\uffcb\063\uffcb\064\uffcb\065\uffcb\066" +
"\uffcb\070\uffcb\001\002\000\046\034\uffd3\036\uffd3\043\uffd3" +
"\047\uffd3\050\uffd3\051\074\052\077\053\073\054\071\057" +
"\uffd3\060\uffd3\061\uffd3\062\uffd3\063\uffd3\064\uffd3\065\uffd3" +
"\066\uffd3\070\uffd3\001\002\000\046\034\uffd1\036\uffd1\043" +
"\uffd1\047\uffd1\050\uffd1\051\uffd1\052\uffd1\053\uffd1\054\071" +
"\057\uffd1\060\uffd1\061\uffd1\062\uffd1\063\uffd1\064\uffd1\065" +
"\uffd1\066\uffd1\070\uffd1\001\002\000\046\034\uffce\036\uffce" +
"\043\uffce\047\uffce\050\uffce\051\uffce\052\uffce\053\uffce\054" +
"\071\057\uffce\060\uffce\061\uffce\062\uffce\063\uffce\064\uffce" +
"\065\uffce\066\uffce\070\uffce\001\002\000\046\034\uffcc\036" +
"\uffcc\043\uffcc\047\075\050\070\051\074\052\077\053\073" +
"\054\071\057\uffcc\060\uffcc\061\uffcc\062\uffcc\063\uffcc\064" +
"\uffcc\065\uffcc\066\uffcc\070\uffcc\001\002\000\044\034\uffcf" +
"\036\uffcf\043\uffcf\047\uffcf\050\uffcf\051\uffcf\052\uffcf\053" +
"\uffcf\057\uffcf\060\uffcf\061\uffcf\062\uffcf\063\uffcf\064\uffcf" +
"\065\uffcf\066\uffcf\070\uffcf\001\002\000\046\034\uffd2\036" +
"\uffd2\043\uffd2\047\uffd2\050\uffd2\051\074\052\077\053\073" +
"\054\071\057\uffd2\060\uffd2\061\uffd2\062\uffd2\063\uffd2\064" +
"\uffd2\065\uffd2\066\uffd2\070\uffd2\001\002\000\046\034\uffc5" +
"\036\uffc5\043\uffc5\047\075\050\070\051\074\052\077\053" +
"\073\054\071\057\101\060\102\061\072\062\065\063\100" +
"\064\076\065\uffc5\066\uffc5\070\uffc5\001\002\000\046\034" +
"\uffc7\036\uffc7\043\uffc7\047\075\050\070\051\074\052\077" +
"\053\073\054\071\057\101\060\102\061\072\062\065\063" +
"\100\064\076\065\uffc7\066\uffc7\070\067\001\002\000\046" +
"\034\uffca\036\uffca\043\uffca\047\075\050\070\051\074\052" +

"\077\053\073\054\071\057\uffca\060\uffca\061\uffca\062\uffca" +
"\063\uffca\064\uffca\065\uffca\066\uffca\070\uffca\001\002\000" +
"\042\002\ufff0\003\ufff0\004\ufff0\012\ufff0\014\ufff0\015\ufff0" +
"\016\ufff0\017\ufff0\020\ufff0\021\ufff0\022\ufff0\023\ufff0\025" +
"\ufff0\026\ufff0\040\ufff0\076\ufff0\001\002\000\046\034\uffd4" +
"\036\uffd4\043\uffd4\047\uffd4\050\uffd4\051\uffd4\052\uffd4\053" +
"\uffd4\054\uffd4\057\uffd4\060\uffd4\061\uffd4\062\uffd4\063\uffd4" +
"\064\uffd4\065\uffd4\066\uffd4\070\uffd4\001\002\000\042\036" +
"\131\047\075\050\070\051\074\052\077\053\073\054\071" +
"\057\101\060\102\061\072\062\065\063\100\064\076\065" +
"\066\066\103\070\067\001\002\000\004\036\127\001\002" +
"\000\026\027\055\030\060\035\051\050\053\067\063\072" +
"\054\073\052\074\062\075\056\076\061\001\002\000\046" +
"\034\uffd5\036\uffd5\043\uffd5\047\075\050\070\051\074\052" +
"\077\053\073\054\071\057\101\060\102\061\072\062\065" +
"\063\100\064\076\065\066\066\103\070\067\001\002\000" +
"\046\034\uffc3\036\uffc3\043\uffc3\047\uffc3\050\uffc3\051\uffc3" +
"\052\uffc3\053\uffc3\054\uffc3\057\uffc3\060\uffc3\061\uffc3\062" +
"\uffc3\063\uffc3\064\uffc3\065\uffc3\066\uffc3\070\uffc3\001\002" +
"\000\004\076\133\001\002\000\004\046\152\001\002\000" +
"\026\027\055\030\060\035\051\050\053\067\063\072\054" +
"\073\052\074\062\075\056\076\061\001\002\000\042\034" +
"\136\047\075\050\070\051\074\052\077\053\073\054\071" +
"\057\101\060\102\061\072\062\065\063\100\064\076\065" +
"\066\066\103\070\067\001\002\000\004\076\137\001\002" +
"\000\010\046\150\055\147\056\146\001\002\000\004\036" +
"\141\001\002\000\004\037\142\001\002\000\032\003\031" +
"\004\035\012\015\014\014\017\010\020\034\021\021\022" +
"\025\023\006\025\030\026\032\076\016\001\002\000\034" +
"\003\031\004\035\012\015\014\014\017\010\020\034\021" +
"\021\022\025\023\006\025\030\026\032\040\145\076\016" +
"\001\002\000\042\002\uffff\003\uffff\004\uffff\012\uffff\014" +
"\uffff\015\uffff\016\uffff\017\uffff\020\uffff\021\uffff\022\uffff" +
"\023\uffff\025\uffff\026\uffff\040\uffff\076\uffff\001\002\000" +
"\042\002\uffe2\003\uffe2\004\uffe2\012\uffe2\014\uffe2\015\uffe2" +
"\016\uffe2\017\uffe2\020\uffe2\021\uffe2\022\uffe2\023\uffe2\025" +
"\uffe2\026\uffe2\040\uffe2\076\uffe2\001\002\000\004\036\uffeb" +
"\001\002\000\004\036\uffec\001\002\000\026\027\055\030" +
"\060\035\051\050\053\067\063\072\054\073\052\074\062" +
"\075\056\076\061\001\002\000\042\036\uffed\047\075\050" +
"\070\051\074\052\077\053\073\054\071\057\101\060\102" +
"\061\072\062\065\063\100\064\076\065\066\066\103\070" +
"\067\001\002\000\026\027\055\030\060\035\051\050\053" +
"\067\063\072\054\073\052\074\062\075\056\076\061\001" +
"\002\000\042\034\154\047\075\050\070\051\074\052\077" +
"\053\073\054\071\057\101\060\102\061\072\062\065\063" +
"\100\064\076\065\066\066\103\070\067\001\002\000\064" +
"\002\uffee\003\uffee\004\uffee\012\uffee\014\uffee\015\uffee\016" +
"\uffee\017\uffee\020\uffee\021\uffee\022\uffee\023\uffee\025\uffee" +
"\026\uffee\027\uffee\030\uffee\035\uffee\040\uffee\050\uffee\067" +
"\uffee\072\uffee\073\uffee\074\uffee\075\uffee\076\uffee\001\002" +
"\000\004\036\156\001\002\000\004\034\157\001\002\000" +
"\042\002\uffdf\003\uffdf\004\uffdf\012\uffdf\014\uffdf\015\uffdf" +
"\016\uffdf\017\uffdf\020\uffdf\021\uffdf\022\uffdf\023\uffdf\025" +
"\uffdf\026\uffdf\040\uffdf\076\uffdf\001\002\000\042\002\ufff1" +
"\003\ufff1\004\ufff1\012\ufff1\014\ufff1\015\ufff1\016\ufff1\017" +
"\ufff1\020\ufff1\021\ufff1\022\ufff1\023\ufff1\025\ufff1\026\ufff1" +

```
"\040\ufff1\076\ufff1\001\002\000\026\027\055\030\060\035" +
"\051\050\053\067\063\072\054\073\052\074\062\075\056" +
"\076\061\001\002\000\042\036\163\047\075\050\070\051" +
"\074\052\077\053\073\054\071\057\101\060\102\061\072" +
"\062\065\063\100\064\076\065\066\066\103\070\067\001" +
"\002\000\004\034\164\001\002\000\042\002\uffea\003\uffea" +
"\004\uffea\012\uffea\014\uffea\015\uffea\016\uffea\017\uffea\020" +
"\uffea\021\uffea\022\uffea\023\uffea\025\uffea\026\uffea\040\uffea" +
"\076\uffea\001\002\000\042\002\uffe1\003\uffe1\004\uffe1\012" +
"\uffe1\014\uffe1\015\uffe1\016\uffe1\017\uffe1\020\uffe1\021\uffe1" +
"\022\uffe1\023\uffe1\025\uffe1\026\uffe1\040\uffe1\076\uffe1\001" +
"\002\000\032\003\031\004\035\012\015\014\014\017\010" +
"\020\034\021\021\022\025\023\006\025\030\026\032\076" +
"\016\001\002\000\034\003\031\004\035\012\015\014\014" +
"\017\010\020\034\021\021\022\025\023\006\025\030\026" +
"\032\040\170\076\016\001\002\000\004\017\171\001\002" +
"\000\004\035\172\001\002\000\026\027\055\030\060\035" +
"\051\050\053\067\063\072\054\073\052\074\062\075\056" +
"\076\061\001\002\000\042\036\174\047\075\050\070\051" +
"\074\052\077\053\073\054\071\057\101\060\102\061\072" +
"\062\065\063\100\064\076\065\066\066\103\070\067\001" +
"\002\000\004\034\175\001\002\000\042\002\uffe3\003\uffe3" +
"\004\uffe3\012\uffe3\014\uffe3\015\uffe3\016\uffe3\017\uffe3\020" +
"\uffe3\021\uffe3\022\uffe3\023\uffe3\025\uffe3\026\uffe3\040\uffe3" +
"\076\uffe3\001\002\000\004\034\201\001\002\000\004\034" +
"\200\001\002\000\042\002\uffe9\003\uffe9\004\uffe9\012\uffe9" +
"\014\uffe9\015\uffe9\016\uffe9\017\uffe9\020\uffe9\021\uffe9\022" +
"\uffe9\023\uffe9\025\uffe9\026\uffe9\040\uffe9\076\uffe9\001\002" +
"\000\042\002\uffe8\003\uffe8\004\uffe8\012\uffe8\014\uffe8\015" +
"\uffe8\016\uffe8\017\uffe8\020\uffe8\021\uffe8\022\uffe8\023\uffe8" +
"\025\uffe8\026\uffe8\040\uffe8\076\uffe8\001\002\000\026\027" +
"\055\030\060\035\051\050\053\067\063\072\054\073\052" +
"\074\062\075\056\076\061\001\002\000\042\036\204\047" +
"\075\050\070\051\074\052\077\053\073\054\071\057\101" +
"\060\102\061\072\062\065\063\100\064\076\065\066\066" +
"\103\070\067\001\002\000\004\037\205\001\002\000\032" +
"\003\031\004\035\012\015\014\014\017\010\020\034\021" +
"\021\022\025\023\006\025\030\026\032\076\016\001\002" +
"\000\034\003\031\004\035\012\015\014\014\017\010\020" +
"\034\021\021\022\025\023\006\025\030\026\032\040\207" +
"\076\016\001\002\000\044\002\uffe7\003\uffe7\004\uffe7\012" +
"\uffe7\013\210\014\uffe7\015\uffe7\016\uffe7\017\uffe7\020\uffe7" +
"\021\uffe7\022\uffe7\023\uffe7\025\uffe7\026\uffe7\040\uffe7\076" +
"\uffe7\001\002\000\006\012\015\037\212\001\002\000\042" +
"\002\uffe5\003\uffe5\004\uffe5\012\uffe5\014\uffe5\015\uffe5\016" +
"\uffe5\017\uffe5\020\uffe5\021\uffe5\022\uffe5\023\uffe5\025\uffe5" +
"\026\uffe5\040\uffe5\076\uffe5\001\002\000\032\003\031\004" +
"\035\012\015\014\014\017\010\020\034\021\021\022\025" +
"\023\006\025\030\026\032\076\016\001\002\000\034\003" +
"\031\004\035\012\015\014\014\017\010\020\034\021\021" +
"\022\025\023\006\025\030\026\032\040\214\076\016\001" +
"\002\000\042\002\uffe6\003\uffe6\004\uffe6\012\uffe6\014\uffe6" +
"\015\uffe6\016\uffe6\017\uffe6\020\uffe6\021\uffe6\022\uffe6\023" +
"\uffe6\025\uffe6\026\uffe6\040\uffe6\076\uffe6\001\002\000\026" +
"\027\055\030\060\035\051\050\053\067\063\072\054\073" +
"\052\074\062\075\056\076\061\001\002\000\042\036\217" +
"\047\075\050\070\051\074\052\077\053\073\054\071\057" +
```

```
  "\101\060\102\061\072\062\065\063\100\064\076\065\066" +
  "\066\103\070\067\001\002\000\004\037\220\001\002\000" +
  "\010\015\uffdb\016\uffdb\040\uffdb\001\002\000\010\015\224" +
  "\016\222\040\223\001\002\000\004\043\230\001\002\000" +
  "\042\002\uffde\003\uffde\004\uffde\012\uffde\014\uffde\015\uffde" +
  "\016\uffde\017\uffde\020\uffde\021\uffde\022\uffde\023\uffde\025" +
  "\uffde\026\uffde\040\uffde\076\uffde\001\002\000\026\027\055" +
  "\030\060\035\051\050\053\067\063\072\054\073\052\074" +
  "\062\075\056\076\061\001\002\000\042\043\226\047\075" +
  "\050\070\051\074\052\077\053\073\054\071\057\101\060" +
  "\102\061\072\062\065\063\100\064\076\065\066\066\103" +
  "\070\067\001\002\000\032\003\031\004\035\012\015\014" +
  "\014\017\010\020\034\021\021\022\025\023\006\025\030" +
  "\026\032\076\016\001\002\000\040\003\031\004\035\012" +
  "\015\014\014\015\uffdd\016\uffdd\017\010\020\034\021\021" +
  "\022\025\023\006\025\030\026\032\040\uffdd\076\016\001" +
  "\002\000\032\003\031\004\035\012\015\014\014\017\010" +
  "\020\034\021\021\022\025\023\006\025\030\026\032\076" +
  "\016\001\002\000\040\003\031\004\035\012\015\014\014" +
  "\015\uffdc\016\uffdc\017\010\020\034\021\021\022\025\023" +
  "\006\025\030\026\032\040\uffdc\076\016\001\002\000\004" +
  "\002\001\001\002\000\026\027\055\030\060\035\051\050" +
  "\053\067\063\072\054\073\052\074\062\075\056\076\061" +
  "\001\002\000\042\036\235\047\075\050\070\051\074\052" +
  "\077\053\073\054\071\057\101\060\102\061\072\062\065" +
  "\063\100\064\076\065\066\066\103\070\067\001\002\000" +
  "\004\037\236\001\002\000\032\003\031\004\035\012\015" +
  "\014\014\017\010\020\034\021\021\022\025\023\006\025" +
  "\030\026\032\076\016\001\002\000\034\003\031\004\035" +
  "\012\015\014\014\017\010\020\034\021\021\022\025\023" +
  "\006\025\030\026\032\040\240\076\016\001\002\000\042" +
  "\002\uffe4\003\uffe4\004\uffe4\012\uffe4\014\uffe4\015\uffe4\016" +
  "\uffe4\017\uffe4\020\uffe4\021\uffe4\022\uffe4\023\uffe4\025\uffe4" +
  "\026\uffe4\040\uffe4\076\uffe4\001\002\000\042\002\uffe0\003" +
  "\uffe0\004\uffe0\012\uffe0\014\uffe0\015\uffe0\016\uffe0\017\uffe0" +
  "\020\uffe0\021\uffe0\022\uffe0\023\uffe0\025\uffe0\026\uffe0\040" +
  "\uffe0\076\uffe0\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\237\000\040\002\011\003\004\004\012\005\021\006" +
  "\016\007\026\010\010\011\035\012\032\013\006\014\022" +
  "\015\017\016\003\017\025\020\023\001\001\000\002\001" +
  "\001\000\034\004\143\005\021\006\016\007\026\010\010" +
  "\011\035\012\032\013\006\014\022\015\017\016\003\017" +
  "\025\020\023\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" +
  "\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\024\042\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\023\056" +
"\001\001\000\002\001\001\000\006\023\124\024\125\001" +
"\001\000\002\001\001\000\004\023\123\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\023\063\001\001\000\002\001\001\000\004\023" +
"\121\001\001\000\004\023\120\001\001\000\004\023\117" +
"\001\001\000\004\023\116\001\001\000\004\023\115\001" +
"\001\000\004\023\114\001\001\000\004\023\113\001\001" +
"\000\004\023\112\001\001\000\004\023\111\001\001\000" +
"\004\023\110\001\001\000\004\023\107\001\001\000\004" +
"\023\106\001\001\000\004\023\105\001\001\000\004\023" +
"\104\001\001\000\004\023\103\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\004\023\127\001\001" +
"\000\002\001\001\000\002\001\001\000\004\006\133\001" +
"\001\000\002\001\001\000\004\023\134\001\001\000\002" +
"\001\001\000\004\022\137\001\001\000\002\001\001\000" +
"\002\001\001\000\002\001\001\000\036\003\142\004\012" +
"\005\021\006\016\007\026\010\010\011\035\012\032\013" +
"\006\014\022\015\017\016\003\017\025\020\023\001\001" +
"\000\034\004\143\005\021\006\016\007\026\010\010\011" +
"\035\012\032\013\006\014\022\015\017\016\003\017\025" +
"\020\023\001\001\000\002\001\001\000\002\001\001\000" +
"\002\001\001\000\002\001\001\000\004\023\150\001\001" +
"\000\002\001\001\000\004\023\152\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\023\161\001" +
"\001\000\002\001\001\000\002\001\001\000\002\001\001" +
"\000\002\001\001\000\036\003\166\004\012\005\021\006" +
"\016\007\026\010\010\011\035\012\032\013\006\014\022" +
"\015\017\016\003\017\025\020\023\001\001\000\034\004" +
"\143\005\021\006\016\007\026\010\010\011\035\012\032" +
"\013\006\014\022\015\017\016\003\017\025\020\023\001" +
"\001\000\002\001\001\000\002\001\001\000\004\023\172" +
"\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\023\202\001\001\000\002\001" +
"\001\000\002\001\001\000\036\003\205\004\012\005\021" +
"\006\016\007\026\010\010\011\035\012\032\013\006\014" +
"\022\015\017\016\003\017\025\020\023\001\001\000\034" +
"\004\143\005\021\006\016\007\026\010\010\011\035\012" +
"\032\013\006\014\022\015\017\016\003\017\025\020\023" +
"\001\001\000\004\010\210\001\001\000" +
"\002\001\001\000\036\003\212\004\012\005\021\006\016" +
"\007\026\010\010\011\035\012\032\013\006\014\022\015" +
"\017\016\003\017\025\020\023\001\001\000\034\004\143" +
"\005\021\006\016\007\026\010\010\011\035\012\032\013" +
"\006\014\022\015\017\016\003\017\025\020\023\001\001" +
"\000\002\001\001\000\004\023\215\001\001\000\002\001" +
"\001\000\002\001\001\000\004\021\220\001\001\000\002" +

```
  "\001\001\000\002\001\001\000\002\001\001\000\004\023" +
  "\224\001\001\000\002\001\001\000\036\003\226\004\012" +
  "\005\021\006\016\007\026\010\010\011\035\012\032\013" +
  "\006\014\022\015\017\016\003\017\025\020\023\001\001" +
  "\000\034\004\143\005\021\006\016\007\026\010\010\011" +
  "\035\012\032\013\006\014\022\015\017\016\003\017\025" +
  "\020\023\001\001\000\036\003\230\004\012\005\021\006" +
  "\016\007\026\010\010\011\035\012\032\013\006\014\022" +
  "\015\017\016\003\017\025\020\023\001\001\000\034\004" +
  "\143\005\021\006\016\007\026\010\010\011\035\012\032" +
  "\013\006\014\022\015\017\016\003\017\025\020\023\001" +
  "\001\000\002\001\001\000\004\023\233\001\001\000\002" +
  "\001\001\000\002\001\001\000\036\003\236\004\012\005" +
  "\021\006\016\007\026\010\010\011\035\012\032\013\006" +
  "\014\022\015\017\016\003\017\025\020\023\001\001\000" +
  "\034\004\143\005\021\006\016\007\026\010\010\011\035" +
  "\012\032\013\006\014\022\015\017\016\003\017\025\020" +
  "\023\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$Parser$actions action_obj;

/** Action encapsulation object initializer. */
protected void init_actions()
  {
    action_obj = new CUP$Parser$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$Parser$do_action(act_num, parser, stack, top);
}

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

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

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



  // Variable global para el AST
```

```
  public LinkedList<Instruccion> AST;

  // --- MANEJO DE ERRORES MEJORADO ---
  public void syntax_error(Symbol s){
     // Si el valor es null, usamos el nombre del terminal (ej: "PTCOMA", "LLAVE_C")
     // s.sym contiene el ID numérico del token, sym.terminalNames lo convierte a texto
     String lexema = s.value != null ? s.value.toString() : sym.terminalNames[s.sym];
     int linea = s.left;
     int col = s.right;

     System.out.println("✕ Error Sintáctico Recuperado: " + lexema + " Línea " + linea + " Col " + col);

     // Guardamos el error en la lista global para el reporte HTML
     Errores.agregar("Sintactico", "No se esperaba el token: " + lexema, linea, col);
  }

  public void unrecovered_syntax_error(Symbol s){
     String lexema = s.value != null ? s.value.toString() : sym.terminalNames[s.sym];
     int linea = s.left;
     int col = s.right;

     System.out.println("✕ Error Sintáctico Fatal: " + lexema + " Línea " + linea + " Col " + col);

     // Guardamos el error fatal
     Errores.agregar("Sintactico Fatal", "Error irrecuperable con: " + lexema, linea, col);
  }


/** Cup generated class to encapsulate user supplied action code.*/
@SuppressWarnings({"rawtypes", "unchecked", "unused"})
class CUP$Parser$actions {
 private final Parser parser;

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

 /** Method 0 with the actual generated action code for actions 0 to 300. */
 public final java_cup.runtime.Symbol CUP$Parser$do_action_part00000000(
  int                        CUP$Parser$act_num,
  java_cup.runtime.lr_parser CUP$Parser$parser,
  java.util.Stack            CUP$Parser$stack,
  int                        CUP$Parser$top)
  throws java.lang.Exception
  {
   /* Symbol object for return from actions */
   java_cup.runtime.Symbol CUP$Parser$result;

   /* select the action based on the action number */
   switch (CUP$Parser$act_num)
    {
     /*. . . . . .*/
     case 0: // $START ::= ini EOF
      {
       Object RESULT =null;
```

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

       /*. . . . . .*/
       case 1: // ini ::= instrucciones
        {
          Object RESULT =null;
                    int aleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int aright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    LinkedList<Instruccion> a = (LinkedList<Instruccion>)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;

  AST = a;
  System.out.println("✓ ¡Análisis Sintáctico Finalizado Correctamente!");

          CUP$Parser$result = parser.getSymbolFactory().newSymbol("ini",0,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
        }
       return CUP$Parser$result;

       /*. . . . . .*/
       case 2: // instrucciones ::= instrucciones instruccion
        {
          LinkedList<Instruccion> RESULT =null;
                    int aleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).left;
                    int aright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).right;
                    LinkedList<Instruccion> a = (LinkedList<Instruccion>)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-1)).value;
                    int bleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int bright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    Instruccion b = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;

              if(b!=null) a.add(b);
              RESULT = a;

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

```
        /*. . . . . . . . . . . . . . . . . . .*/
        case 3: // instrucciones ::= instruccion
         {
           LinkedList<Instruccion> RESULT =null;
                        int aleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                        int aright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                        Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;

                LinkedList<Instruccion> lista = new LinkedList<>();
                if(a!=null) lista.add(a);
                RESULT = lista;

           CUP$Parser$result = parser.getSymbolFactory().newSymbol("instrucciones",1,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . . . . . . . . . . . . . . .*/
        case 4: // instruccion ::= declaracion
         {
           Instruccion RESULT =null;
                        int aleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                        int aright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                        Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                         RESULT = a;
           CUP$Parser$result = parser.getSymbolFactory().newSymbol("instruccion",2,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . . . . . . . . . . . . . . .*/
        case 5: // instruccion ::= asignacion
         {
           Instruccion RESULT =null;
                        int aleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                        int aright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                        Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                         RESULT = a;
           CUP$Parser$result = parser.getSymbolFactory().newSymbol("instruccion",2,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . . . . . . . . . . . . . . .*/
        case 6: // instruccion ::= impresion
         {
           Instruccion RESULT =null;
                        int aleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                        int aright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                        Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
```

```
                    RESULT = a;
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("instruccion",2,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
        }
       return CUP$Parser$result;

       /*. . . . . .*/
       case 7: // instruccion ::= if_st
        {
          Instruccion RESULT =null;
                    int aleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int aright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                     RESULT = a;
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("instruccion",2,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
        }
       return CUP$Parser$result;

       /*. . . . . .*/
       case 8: // instruccion ::= while_st
        {
          Instruccion RESULT =null;
                    int aleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int aright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                     RESULT = a;
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("instruccion",2,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
        }
       return CUP$Parser$result;

       /*. . . . . .*/
       case 9: // instruccion ::= do_while_st
        {
          Instruccion RESULT =null;
                    int aleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int aright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                     RESULT = a;
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("instruccion",2,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
        }
       return CUP$Parser$result;

       /*. . . . . .*/
       case 10: // instruccion ::= for_st
        {
          Instruccion RESULT =null;
                    int aleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
```

```
                    int aright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                     RESULT = a;
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("instruccion",2,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 11: // instruccion ::= switch_st
         {
          Instruccion RESULT =null;
                    int aleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int aright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                     RESULT = a;
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("instruccion",2,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 12: // instruccion ::= start_st
         {
          Instruccion RESULT =null;
                    int aleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int aright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                     RESULT = a;
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("instruccion",2,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 13: // instruccion ::= break_st
         {
          Instruccion RESULT =null;
                    int aleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int aright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                     RESULT = a;
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("instruccion",2,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 14: // instruccion ::= continue_st
```

```
        {
          Instruccion RESULT =null;
                    int aleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int aright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                     RESULT = a;
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("instruccion",2,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
        }
       return CUP$Parser$result;

       /*. . . . . .*/
       case 15: // instruccion ::= incremento_st
        {
          Instruccion RESULT =null;
                    int aleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int aright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                     RESULT = a;
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("instruccion",2,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
        }
       return CUP$Parser$result;

       /*. . . . . .*/
       case 16: // instruccion ::= error PTCOMA
        {
          Instruccion RESULT =null;
                     System.out.println("Error recuperado"); RESULT = null;
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("instruccion",2,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
        }
       return CUP$Parser$result;

       /*. . . . . .*/
       case 17: // declaracion ::= VAR ID DOSPTOS tipo IGUAL expresion PTCOMA
        {
          Instruccion RESULT =null;
                    int idleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-5)).left;
                    int idright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-5)).right;
                    String id = (String)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-5)).value;
                    int tleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-3)).left;
                    int tright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-3)).right;
                    String t = (String)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-3)).value;
                    int eleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).left;
```

```
                    int eright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).right;
                    Instruccion e = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-1)).value;
                     RESULT = new Declaracion(id, t, e, idleft, idright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("declaracion",3,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-6)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 18: // declaracion ::= VAR ID DOSPTOS tipo PTCOMA
         {
           Instruccion RESULT =null;
                    int idleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-3)).left;
                    int idright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-3)).right;
                    String id = (String)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-3)).value;
                    int tleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).left;
                    int tright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).right;
                    String t = (String)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-1)).value;
                     RESULT = new Declaracion(id, t, null, idleft, idright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("declaracion",3,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-4)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 19: // asignacion ::= ID IGUAL expresion PTCOMA
         {
           Instruccion RESULT =null;
                    int idleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-3)).left;
                    int idright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-3)).right;
                    String id = (String)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-3)).value;
                    int eleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).left;
                    int eright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).right;
                    Instruccion e = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-1)).value;
                     RESULT = new Asignacion(id, e, idleft, idright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("asignacion",4,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-3)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;
```

```
        /*. . . . . . . . . . . . . . . . . . .*/
        case 20: // asignacion_for ::= ID IGUAL expresion
         {
           Instruccion RESULT =null;
                        int idleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).left;
                        int idright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).right;
                        String id = (String)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-2)).value;
                        int eleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                        int eright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                        Instruccion e = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                         RESULT = new Asignacion(id, e, idleft, idright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("asignacion_for",16,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . . . . . . . . . . . . . . .*/
        case 21: // asignacion_for ::= ID MASMAS
         {
           Instruccion RESULT =null;
                        int idleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).left;
                        int idright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).right;
                        String id = (String)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-1)).value;
                         RESULT = new Incremento(id, true, idleft, idright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("asignacion_for",16,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . . . . . . . . . . . . . . .*/
        case 22: // asignacion_for ::= ID MENOSMENOS
         {
           Instruccion RESULT =null;
                        int idleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).left;
                        int idright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).right;
                        String id = (String)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-1)).value;
                         RESULT = new Incremento(id, false, idleft, idright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("asignacion_for",16,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . . . . . . . . . . . . . . .*/
        case 23: // impresion ::= PRINT PAR_A expresion PAR_C PTCOMA
```

```
        {
          Instruccion RESULT =null;
                    int eleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).left;
                    int eright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).right;
                    Instruccion e = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-2)).value;
                     RESULT = new Impresion(e, eleft, eright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("impresion",5,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-4)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
        }
      return CUP$Parser$result;

      /*. . . . . .*/
      case 24: // incremento_st ::= ID MASMAS PTCOMA
        {
          Instruccion RESULT =null;
                    int idleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).left;
                    int idright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).right;
                    String id = (String)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-2)).value;
                     RESULT = new Incremento(id, true, idleft, idright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("incremento_st",14,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
        }
      return CUP$Parser$result;

      /*. . . . . .*/
      case 25: // incremento_st ::= ID MENOSMENOS PTCOMA
        {
          Instruccion RESULT =null;
                    int idleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).left;
                    int idright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).right;
                    String id = (String)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-2)).value;
                     RESULT = new Incremento(id, false, idleft, idright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("incremento_st",14,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
        }
      return CUP$Parser$result;

      /*. . . . . .*/
      case 26: // if_st ::= IF PAR_A expresion PAR_C LLAVE_A instrucciones LLAVE_C
        {
          Instruccion RESULT =null;
                    int pleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-6)).left;
                    int pright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-6)).right;
```

```
                    String p = (String)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-6)).value;
                    int condleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-4)).left;
                    int condright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-4)).right;
                    Instruccion cond = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-4)).value;
                    int bloqueleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).left;
                    int bloqueright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).right;
                    LinkedList<Instruccion> bloque =
(LinkedList<Instruccion>)((java_cup.runtime.Symbol) CUP$Parser$stack.elementAt(CUP$Parser$top-
1)).value;
                     RESULT = new If(cond, bloque, null, pleft, pright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("if_st",6,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-6)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 27: // if_st ::= IF PAR_A expresion PAR_C LLAVE_A instrucciones LLAVE_C ELSE LLAVE_A
instrucciones LLAVE_C
         {
           Instruccion RESULT =null;
                    int pleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-10)).left;
                    int pright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-10)).right;
                    String p = (String)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-10)).value;
                    int condleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-8)).left;
                    int condright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-8)).right;
                    Instruccion cond = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-8)).value;
                    int b1left =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-5)).left;
                    int b1right =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-5)).right;
                    LinkedList<Instruccion> b1 = (LinkedList<Instruccion>)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-5)).value;
                    int b2left =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).left;
                    int b2right =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).right;
                    LinkedList<Instruccion> b2 = (LinkedList<Instruccion>)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-1)).value;
                     RESULT = new If(cond, b1, b2, pleft, pright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("if_st",6,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-10)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;
```

```
        /*. . . . . . . . . . . . . . . . . . .*/
        case 28: // if_st ::= IF PAR_A expresion PAR_C LLAVE_A instrucciones LLAVE_C ELSE if_st
         {
           Instruccion RESULT =null;
                        int pleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-8)).left;
                        int pright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-8)).right;
                        String p = (String)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-8)).value;
                        int condleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-6)).left;
                        int condright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-6)).right;
                        Instruccion cond = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-6)).value;
                        int b1left =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-3)).left;
                        int b1right =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-3)).right;
                        LinkedList<Instruccion> b1 = (LinkedList<Instruccion>)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-3)).value;
                        int b2left = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                        int b2right = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                        Instruccion b2 = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;

          // Regla especial para "else if"
          LinkedList<Instruccion> listaElseIf = new LinkedList<>();
          listaElseIf.add(b2);
          RESULT = new If(cond, b1, listaElseIf, pleft, pright);

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

        /*. . . . . . . . . . . . . . . . . . .*/
        case 29: // while_st ::= WHILE PAR_A expresion PAR_C LLAVE_A instrucciones LLAVE_C
         {
           Instruccion RESULT =null;
                        int pleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-6)).left;
                        int pright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-6)).right;
                        String p = (String)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-6)).value;
                        int condleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-4)).left;
                        int condright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-4)).right;
                        Instruccion cond = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-4)).value;
                        int bloqueleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).left;
```

```
                    int bloqueright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).right;
                    LinkedList<Instruccion> bloque =
(LinkedList<Instruccion>)((java_cup.runtime.Symbol) CUP$Parser$stack.elementAt(CUP$Parser$top-
1)).value;
                     RESULT = new While(cond, bloque, pleft, right);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("while_st",7,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-6)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 30: // do_while_st ::= DO LLAVE_A instrucciones LLAVE_C WHILE PAR_A expresion PAR_C
PTCOMA
         {
           Instruccion RESULT =null;
                    int pleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-8)).left;
                    int pright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-8)).right;
                    String p = (String)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-8)).value;
                    int bloqueleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-6)).left;
                    int bloqueright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-6)).right;
                    LinkedList<Instruccion> bloque =
(LinkedList<Instruccion>)((java_cup.runtime.Symbol) CUP$Parser$stack.elementAt(CUP$Parser$top-
6)).value;
                    int condleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).left;
                    int condright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).right;
                    Instruccion cond = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-2)).value;
                     RESULT = new DoWhile(cond, bloque, pleft, right);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("do_while_st",8,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-8)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 31: // for_st ::= FOR PAR_A asignacion expresion PTCOMA asignacion_for PAR_C LLAVE_A
instrucciones LLAVE_C
         {
           Instruccion RESULT =null;
                    int pleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-9)).left;
                    int pright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-9)).right;
                    String p = (String)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-9)).value;
                    int asigleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-7)).left;
```

```
                    int asigright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-7)).right;
                    Instruccion asig = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-7)).value;
                    int condleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-6)).left;
                    int condright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-6)).right;
                    Instruccion cond = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-6)).value;
                    int actleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-4)).left;
                    int actright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-4)).right;
                    Instruccion act = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-4)).value;
                    int bloqueleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).left;
                    int bloqueright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).right;
                    LinkedList<Instruccion> bloque =
(LinkedList<Instruccion>)((java_cup.runtime.Symbol) CUP$Parser$stack.elementAt(CUP$Parser$top-
1)).value;
                     RESULT = new For(asig, cond, act, bloque, pleft, pright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("for_st",9,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-9)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 32: // break_st ::= BREAK PTCOMA
         {
           Instruccion RESULT =null;
                    int pleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).left;
                    int pright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).right;
                    String p = (String)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-1)).value;
                     RESULT = new Break(pleft, pright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("break_st",11,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 33: // continue_st ::= CONTINUE PTCOMA
         {
           Instruccion RESULT =null;
                    int pleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).left;
                    int pright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).right;
                    String p = (String)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-1)).value;
```

```
                    RESULT = new Continue(pleft, pright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("continue_st",12,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
        }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 34: // start_st ::= START PAR_A PAR_C PTCOMA
         {
           Instruccion RESULT =null;
                    int pleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-3)).left;
                    int pright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-3)).right;
                    String p = (String)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-3)).value;
                     System.out.println("START detectado"); RESULT = null;
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("start_st",13,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-3)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
        }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 35: // switch_st ::= SWITCH PAR_A expresion PAR_C LLAVE_A CaseList LLAVE_C
         {
           Instruccion RESULT =null;
                    int pleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-6)).left;
                    int pright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-6)).right;
                    String p = (String)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-6)).value;
                    int eleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-4)).left;
                    int eright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-4)).right;
                    Instruccion e = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-4)).value;
                    int cleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).left;
                    int cright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).right;
                    LinkedList<Case> c = (LinkedList<Case>)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-1)).value;
                     RESULT = new Switch(e, c, pleft, pright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("switch_st",10,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-6)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
        }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 36: // CaseList ::= CaseList CASE expresion DOSPTOS instrucciones
         {
           LinkedList<Case> RESULT =null;
```

```
                    int lleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-
4)).left;
                    int lright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-4)).right;
                    LinkedList<Case> l = (LinkedList<Case>)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-4)).value;
                    int eleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).left;
                    int eright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).right;
                    Instruccion e = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-2)).value;
                    int bloqueleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int bloqueright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    LinkedList<Instruccion> bloque =
(LinkedList<Instruccion>)((java_cup.runtime.Symbol) CUP$Parser$stack.peek()).value;

          l.add(new Case(e, bloque, false));
          RESULT = l;

         CUP$Parser$result = parser.getSymbolFactory().newSymbol("CaseList",15,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-4)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
        }
      return CUP$Parser$result;

      /*. . . . . .*/
      case 37: // CaseList ::= CaseList DEFAULT DOSPTOS instrucciones
       {
         LinkedList<Case> RESULT =null;
                    int lleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-
3)).left;
                    int lright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-3)).right;
                    LinkedList<Case> l = (LinkedList<Case>)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-3)).value;
                    int bloqueleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int bloqueright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    LinkedList<Instruccion> bloque =
(LinkedList<Instruccion>)((java_cup.runtime.Symbol) CUP$Parser$stack.peek()).value;

          l.add(new Case(null, bloque, true));
          RESULT = l;

         CUP$Parser$result = parser.getSymbolFactory().newSymbol("CaseList",15,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-3)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
        }
      return CUP$Parser$result;

      /*. . . . . .*/
      case 38: // CaseList ::=
       {
         LinkedList<Case> RESULT =null;

          RESULT = new LinkedList<Case>();
```

```
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("CaseList",15,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 39: // tipo ::= INT
         {
          String RESULT =null;
                    RESULT = "int";
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("tipo",18,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 40: // tipo ::= DOUBLE
         {
          String RESULT =null;
                    RESULT = "double";
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("tipo",18,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 41: // tipo ::= BOOL
         {
          String RESULT =null;
                    RESULT = "bool";
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("tipo",18,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 42: // tipo ::= CHAR
         {
          String RESULT =null;
                    RESULT = "char";
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("tipo",18,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 43: // tipo ::= STRING
         {
          String RESULT =null;
                    RESULT = "string";
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("tipo",18,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
```

```
         }
        return CUP$Parser$result;

        /*. . . . . . . . . . . . . . . . . . .*/
        case 44: // expresion ::= PAR_A tipo PAR_C expresion
         {
           Instruccion RESULT =null;
                        int tleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).left;
                        int tright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).right;
                        String t = (String)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-2)).value;
                        int eleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                        int eright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                        Instruccion e = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                         RESULT = new Casteo(t, e, tleft, tright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("expresion",17,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-3)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . . . . . . . . . . . . . . .*/
        case 45: // expresion ::= MENOS expresion
         {
           Instruccion RESULT =null;
                        int aleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                        int aright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                        Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                         RESULT = new Aritmetica(a, Operacion.NEGACION, aleft, aright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("expresion",17,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . . . . . . . . . . . . . . .*/
        case 46: // expresion ::= expresion MAS expresion
         {
           Instruccion RESULT =null;
                        int aleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).left;
                        int aright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).right;
                        Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-2)).value;
                        int bleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                        int bright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                        Instruccion b = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                         RESULT = new Aritmetica(a, b, Operacion.SUMA, aleft, aright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("expresion",17,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
```

```
         }
        return CUP$Parser$result;

        /*. . . . . . . . . . . . . . . . . . .*/
        case 47: // expresion ::= expresion MENOS expresion
         {
           Instruccion RESULT =null;
                    int aleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).left;
                    int aright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).right;
                    Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-2)).value;
                    int bleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int bright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    Instruccion b = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                     RESULT = new Aritmetica(a, b, Operacion.RESTA, aleft, aright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("expresion",17,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . . . . . . . . . . . . . . .*/
        case 48: // expresion ::= expresion POR expresion
         {
           Instruccion RESULT =null;
                    int aleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).left;
                    int aright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).right;
                    Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-2)).value;
                    int bleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int bright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    Instruccion b = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                     RESULT = new Aritmetica(a, b, Operacion.MULTIPLICACION, aleft, aright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("expresion",17,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . . . . . . . . . . . . . . .*/
        case 49: // expresion ::= expresion DIV expresion
         {
           Instruccion RESULT =null;
                    int aleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).left;
                    int aright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).right;
                    Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-2)).value;
                    int bleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int bright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
```

```
                    Instruccion b = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                     RESULT = new Aritmetica(a, b, Operacion.DIVISION, aleft, aright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("expresion",17,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 50: // expresion ::= expresion POTENCIA expresion
         {
           Instruccion RESULT =null;
                    int aleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).left;
                    int aright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).right;
                    Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-2)).value;
                    int bleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int bright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    Instruccion b = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                     RESULT = new Aritmetica(a, b, Operacion.POTENCIA, aleft, aright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("expresion",17,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 51: // expresion ::= expresion MOD expresion
         {
           Instruccion RESULT =null;
                    int aleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).left;
                    int aright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).right;
                    Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-2)).value;
                    int bleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int bright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    Instruccion b = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                     RESULT = new Aritmetica(a, b, Operacion.MODULO, aleft, aright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("expresion",17,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 52: // expresion ::= expresion MAYOR expresion
         {
           Instruccion RESULT =null;
                    int aleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).left;
```

```
                    int aright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).right;
                    Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-2)).value;
                    int bleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int bright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    Instruccion b = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                     RESULT = new Logica(a, b, Operacion.MAYOR, aleft, aright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("expresion",17,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 53: // expresion ::= expresion MENOR expresion
         {
           Instruccion RESULT =null;
                    int aleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).left;
                    int aright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).right;
                    Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-2)).value;
                    int bleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int bright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    Instruccion b = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                     RESULT = new Logica(a, b, Operacion.MENOR, aleft, aright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("expresion",17,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 54: // expresion ::= expresion MAYORIGUAL expresion
         {
           Instruccion RESULT =null;
                    int aleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).left;
                    int aright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).right;
                    Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-2)).value;
                    int bleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int bright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    Instruccion b = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                     RESULT = new Logica(a, b, Operacion.MAYORIGUAL, aleft, aright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("expresion",17,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;
```

```
        /*. . . . . . . . . . . . . . . . . . .*/
        case 55: // expresion ::= expresion MENORIGUAL expresion
         {
           Instruccion RESULT =null;
                    int aleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).left;
                    int aright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).right;
                    Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-2)).value;
                    int bleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int bright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    Instruccion b = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                     RESULT = new Logica(a, b, Operacion.MENORIGUAL, aleft, aright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("expresion",17,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . . . . . . . . . . . . . . .*/
        case 56: // expresion ::= expresion IGUALIGUAL expresion
         {
           Instruccion RESULT =null;
                    int aleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).left;
                    int aright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).right;
                    Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-2)).value;
                    int bleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int bright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    Instruccion b = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                     RESULT = new Logica(a, b, Operacion.IGUALIGUAL, aleft, aright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("expresion",17,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . . . . . . . . . . . . . . .*/
        case 57: // expresion ::= expresion DIFERENTE expresion
         {
           Instruccion RESULT =null;
                    int aleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).left;
                    int aright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).right;
                    Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-2)).value;
                    int bleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int bright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    Instruccion b = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                     RESULT = new Logica(a, b, Operacion.DIFERENTE, aleft, aright);
```

```
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("expresion",17,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
        }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 58: // expresion ::= expresion AND expresion
         {
           Instruccion RESULT =null;
                    int aleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).left;
                    int aright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).right;
                    Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-2)).value;
                    int bleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int bright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    Instruccion b = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                     RESULT = new Logica(a, b, Operacion.AND, aleft, aright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("expresion",17,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
        }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 59: // expresion ::= expresion OR expresion
         {
           Instruccion RESULT =null;
                    int aleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).left;
                    int aright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).right;
                    Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-2)).value;
                    int bleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int bright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    Instruccion b = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                     RESULT = new Logica(a, b, Operacion.OR, aleft, aright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("expresion",17,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
        }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 60: // expresion ::= expresion XOR expresion
         {
           Instruccion RESULT =null;
                    int aleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).left;
                    int aright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)).right;
```

```
                    Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-2)).value;
                    int bleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int bright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    Instruccion b = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                     RESULT = new Logica(a, b, Operacion.XOR, aleft, aright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("expresion",17,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
        }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 61: // expresion ::= NOT expresion
         {
           Instruccion RESULT =null;
                    int aleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int aright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.peek()).value;
                     RESULT = new Logica(a, Operacion.NOT, aleft, aright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("expresion",17,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
        }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 62: // expresion ::= PAR_A expresion PAR_C
         {
           Instruccion RESULT =null;
                    int aleft =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).left;
                    int aright =
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-1)).right;
                    Instruccion a = (Instruccion)((java_cup.runtime.Symbol)
CUP$Parser$stack.elementAt(CUP$Parser$top-1)).value;
                     RESULT = a;
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("expresion",17,
((java_cup.runtime.Symbol)CUP$Parser$stack.elementAt(CUP$Parser$top-2)),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
        }
        return CUP$Parser$result;

        /*. . . . . .*/
        case 63: // expresion ::= ENTERO
         {
           Instruccion RESULT =null;
                    int aleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                    int aright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                    String a = (String)((java_cup.runtime.Symbol) CUP$Parser$stack.peek()).value;
                     RESULT = new Primitivo(Double.parseDouble(a), aleft, aright);
          CUP$Parser$result = parser.getSymbolFactory().newSymbol("expresion",17,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
        }
```

```
        return CUP$Parser$result;

        /*. . . . . . . . . . . . . . . . . . .*/
        case 64: // expresion ::= DECIMAL
         {
           Instruccion RESULT =null;
                        int aleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                        int aright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                        String a = (String)((java_cup.runtime.Symbol) CUP$Parser$stack.peek()).value;
                         RESULT = new Primitivo(Double.parseDouble(a), aleft, aright);
           CUP$Parser$result = parser.getSymbolFactory().newSymbol("expresion",17,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . . . . . . . . . . . . . . .*/
        case 65: // expresion ::= LIT_STRING
         {
           Instruccion RESULT =null;
                        int aleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                        int aright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                        String a = (String)((java_cup.runtime.Symbol) CUP$Parser$stack.peek()).value;
                         RESULT = new Primitivo(a, aleft, aright);
           CUP$Parser$result = parser.getSymbolFactory().newSymbol("expresion",17,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . . . . . . . . . . . . . . .*/
        case 66: // expresion ::= LIT_CHAR
         {
           Instruccion RESULT =null;
                        int aleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                        int aright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                        String a = (String)((java_cup.runtime.Symbol) CUP$Parser$stack.peek()).value;
                         RESULT = new Primitivo(a, aleft, aright);
           CUP$Parser$result = parser.getSymbolFactory().newSymbol("expresion",17,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . . . . . . . . . . . . . . .*/
        case 67: // expresion ::= TRUE
         {
           Instruccion RESULT =null;
                        int aleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                        int aright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                        String a = (String)((java_cup.runtime.Symbol) CUP$Parser$stack.peek()).value;
                         RESULT = new Primitivo(true, aleft, aright);
           CUP$Parser$result = parser.getSymbolFactory().newSymbol("expresion",17,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;
```

```
        /*. . . . . . . . . . . . . . . . . . .*/
        case 68: // expresion ::= FALSE
         {
           Instruccion RESULT =null;
                        int aleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                        int aright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                        String a = (String)((java_cup.runtime.Symbol) CUP$Parser$stack.peek()).value;
                         RESULT = new Primitivo(false, aleft, aright);
           CUP$Parser$result = parser.getSymbolFactory().newSymbol("expresion",17,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

        /*. . . . . . . . . . . . . . . . . . .*/
        case 69: // expresion ::= ID
         {
           Instruccion RESULT =null;
                        int aleft = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).left;
                        int aright = ((java_cup.runtime.Symbol)CUP$Parser$stack.peek()).right;
                        String a = (String)((java_cup.runtime.Symbol) CUP$Parser$stack.peek()).value;
                         RESULT = new AccesoVar(a, aleft, aright);
           CUP$Parser$result = parser.getSymbolFactory().newSymbol("expresion",17,
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()),
((java_cup.runtime.Symbol)CUP$Parser$stack.peek()), RESULT);
         }
        return CUP$Parser$result;

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

      }
    } /* end of method */

  /** Method splitting the generated action code into several parts. */
  public final java_cup.runtime.Symbol CUP$Parser$do_action(
    int                        CUP$Parser$act_num,
    java_cup.runtime.lr_parser CUP$Parser$parser,
    java.util.Stack            CUP$Parser$stack,
    int                        CUP$Parser$top)
    throws java.lang.Exception
    {
           return CUP$Parser$do_action_part00000000(
                        CUP$Parser$act_num,
                        CUP$Parser$parser,
                        CUP$Parser$stack,
                        CUP$Parser$top);
    }
}

}
```