```
/* 1. CONFIGURACIÓN E IMPORTACIONES */
package analizadores;
import java_cup.runtime.*;
import java.util.LinkedList;
import arbol.*;
import excepciones.Errores;

parser code {:
   // 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);
   }
:}

/* 2. TERMINALES */
terminal String VAR, INT, DOUBLE, BOOL, CHAR, STRING;
terminal String IF, ELSE, SWITCH, CASE, DEFAULT, WHILE, FOR, DO;
terminal String BREAK, CONTINUE, RETURN, PRINT, START;
terminal String TRUE, FALSE, NEW, LIST, APPEND;

terminal String PTCOMA, PAR_A, PAR_C, LLAVE_A, LLAVE_C, COR_IZQ, COR_DER;
terminal String DOSPTOS, COMA, PUNTO;
terminal String IGUAL;

terminal String MAS, MENOS, POR, DIV, MOD, POTENCIA, MASMAS, MENOSMENOS;
terminal String IGUALIGUAL, DIFERENTE, MENOR, MENORIGUAL, MAYOR, MAYORIGUAL;
terminal String AND, OR, NOT, XOR;
terminal UMENOS;

terminal String ENTERO, DECIMAL, LIT_STRING, LIT_CHAR, ID;

/* 3. NO TERMINALES */
non terminal ini;
non terminal LinkedList<Instruccion> instrucciones;
```

```
non terminal Instruccion instruccion;

non terminal Instruccion declaracion, asignacion, impresion;
non terminal Instruccion if_st, while_st, do_while_st, for_st, switch_st;
non terminal Instruccion break_st, continue_st, start_st;
non terminal Instruccion incremento_st;

non terminal LinkedList<Case> CaseList;
non terminal Instruccion asignacion_for;
non terminal Instruccion expresion;
non terminal String tipo;

/* 4. PRECEDENCIA */
precedence left OR;
precedence left AND;
precedence left XOR;
precedence right NOT;
precedence left IGUALIGUAL, DIFERENTE, MENOR, MENORIGUAL, MAYOR, MAYORIGUAL;
precedence left MAS, MENOS;
precedence left POR, DIV, MOD;
precedence nonassoc POTENCIA;
precedence right UMENOS;

/* 5. GRAMÁTICA */

start with ini;

ini ::= instrucciones:a {:
   AST = a;
   System.out.println("✅ ¡Análisis Sintáctico Finalizado Correctamente!");
:};

instrucciones ::= instrucciones:a instruccion:b {:
                    if(b!=null) a.add(b);
                    RESULT = a;
                  :}
                  |
                  instruccion:a {:
                    LinkedList<Instruccion> lista = new LinkedList<>();
                    if(a!=null) lista.add(a);
                    RESULT = lista;
                  :};

instruccion ::= declaracion:a   {: RESULT = a; :}
              | asignacion:a    {: RESULT = a; :}
              | impresion:a     {: RESULT = a; :}
              | if_st:a         {: RESULT = a; :}
              | while_st:a      {: RESULT = a; :}
              | do_while_st:a   {: RESULT = a; :}
              | for_st:a        {: RESULT = a; :}
              | switch_st:a     {: RESULT = a; :}
              | start_st:a      {: RESULT = a; :}
              | break_st:a      {: RESULT = a; :}
              | continue_st:a   {: RESULT = a; :}
              | incremento_st:a {: RESULT = a; :}
              | error PTCOMA    {: System.out.println("Error recuperado"); RESULT = null; :};
```

```
/* SENTENCIAS */
declaracion ::= VAR ID:id DOSPTOS tipo:t IGUAL expresion:e PTCOMA {: RESULT = new Declaracion(id,
t, e, idleft, idright); :}
          | VAR ID:id DOSPTOS tipo:t PTCOMA {: RESULT = new Declaracion(id, t, null, idleft, idright); :};
// NUEVA REGLA PARA DECLARAR SIN VALOR

asignacion ::= ID:id IGUAL expresion:e PTCOMA {: RESULT = new Asignacion(id, e, idleft, idright); :};

asignacion_for ::= ID:id IGUAL expresion:e {: RESULT = new Asignacion(id, e, idleft, idright); :}
            | ID:id MASMAS {: RESULT = new Incremento(id, true, idleft, idright); :}
            | ID:id MENOSMENOS {: RESULT = new Incremento(id, false, idleft, idright); :};

impresion ::= PRINT PAR_A expresion:e PAR_C PTCOMA {: RESULT = new Impresion(e, eleft, eright); :};

/* INCREMENTO / DECREMENTO */
incremento_st ::= ID:id MASMAS PTCOMA {: RESULT = new Incremento(id, true, idleft, idright); :}
           | ID:id MENOSMENOS PTCOMA {: RESULT = new Incremento(id, false, idleft, idright); :};

/* CONTROL */
if_st ::= IF:p PAR_A expresion:cond PAR_C LLAVE_A instrucciones:bloque LLAVE_C {: RESULT = new
If(cond, bloque, null, pleft, pright); :}
      | IF:p PAR_A expresion:cond PAR_C LLAVE_A instrucciones:b1 LLAVE_C ELSE LLAVE_A
instrucciones:b2 LLAVE_C {: RESULT = new If(cond, b1, b2, pleft, pright); :}
      | IF:p PAR_A expresion:cond PAR_C LLAVE_A instrucciones:b1 LLAVE_C ELSE if_st:b2 {:
          // Regla especial para "else if"
          LinkedList<Instruccion> listaElseIf = new LinkedList<>();
          listaElseIf.add(b2);
          RESULT = new If(cond, b1, listaElseIf, pleft, pright);
        :};

while_st ::= WHILE:p PAR_A expresion:cond PAR_C LLAVE_A instrucciones:bloque LLAVE_C {: RESULT
= new While(cond, bloque, pleft, pright); :};

do_while_st ::= DO:p LLAVE_A instrucciones:bloque LLAVE_C WHILE PAR_A expresion:cond PAR_C
PTCOMA {: RESULT = new DoWhile(cond, bloque, pleft, pright); :};

for_st ::= FOR:p PAR_A asignacion:asig expresion:cond PTCOMA asignacion_for:act PAR_C LLAVE_A
instrucciones:bloque LLAVE_C {: RESULT = new For(asig, cond, act, bloque, pleft, pright); :};

break_st ::= BREAK:p PTCOMA {: RESULT = new Break(pleft, pright); :};
continue_st ::= CONTINUE:p PTCOMA {: RESULT = new Continue(pleft, pright); :};
start_st ::= START:p PAR_A PAR_C PTCOMA {: System.out.println("START detectado"); RESULT =
null; :};

/* SWITCH */
switch_st ::= SWITCH:p PAR_A expresion:e PAR_C LLAVE_A CaseList:c LLAVE_C {: RESULT = new
Switch(e, c, pleft, pright); :};

CaseList ::= CaseList:l CASE expresion:e DOSPTOS instrucciones:bloque {:
           l.add(new Case(e, bloque, false));
           RESULT = l;
         :}
         | CaseList:l DEFAULT DOSPTOS instrucciones:bloque {:
           l.add(new Case(null, bloque, true));
           RESULT = l;
         :}
         | /* vacio */ {:
```

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

/* EXPRESIONES */
tipo ::= INT {: RESULT = "int"; :} | DOUBLE {: RESULT = "double"; :} |
       BOOL {: RESULT = "bool"; :} | CHAR {: RESULT = "char"; :} | STRING {: RESULT = "string"; :};

expresion ::=
         // Regla de Casteo
          PAR_A tipo:t PAR_C expresion:e    {: RESULT = new Casteo(t, e, tleft, tright); :}

         | MENOS expresion:a {: RESULT = new Aritmetica(a, Operacion.NEGACION, aleft,
aright); :}%prec UMENOS

         | expresion:a MAS expresion:b     {: RESULT = new Aritmetica(a, b, Operacion.SUMA, aleft,
aright); :}
         | expresion:a MENOS expresion:b   {: RESULT = new Aritmetica(a, b, Operacion.RESTA, aleft,
aright); :}
         | expresion:a POR expresion:b     {: RESULT = new Aritmetica(a, b, Operacion.MULTIPLICACION,
aleft, aright); :}
         | expresion:a DIV expresion:b     {: RESULT = new Aritmetica(a, b, Operacion.DIVISION, aleft,
aright); :}
         | expresion:a POTENCIA expresion:b {: RESULT = new Aritmetica(a, b, Operacion.POTENCIA,
aleft, aright); :}
         | expresion:a MOD expresion:b     {: RESULT = new Aritmetica(a, b, Operacion.MODULO, aleft,
aright); :}

         | expresion:a MAYOR expresion:b     {: RESULT = new Logica(a, b, Operacion.MAYOR, aleft,
aright); :}
         | expresion:a MENOR expresion:b     {: RESULT = new Logica(a, b, Operacion.MENOR, aleft,
aright); :}
         | expresion:a MAYORIGUAL expresion:b {: RESULT = new Logica(a, b, Operacion.MAYORIGUAL,
aleft, aright); :}
         | expresion:a MENORIGUAL expresion:b {: RESULT = new Logica(a, b, Operacion.MENORIGUAL,
aleft, aright); :}
         | expresion:a IGUALIGUAL expresion:b {: RESULT = new Logica(a, b, Operacion.IGUALIGUAL,
aleft, aright); :}
         | expresion:a DIFERENTE expresion:b  {: RESULT = new Logica(a, b, Operacion.DIFERENTE, aleft,
aright); :}

         | expresion:a AND expresion:b      {: RESULT = new Logica(a, b, Operacion.AND, aleft, aright); :}
         | expresion:a OR expresion:b       {: RESULT = new Logica(a, b, Operacion.OR, aleft, aright); :}
         | expresion:a XOR expresion:b      {: RESULT = new Logica(a, b, Operacion.XOR, aleft, aright); :}
         | NOT expresion:a                  {: RESULT = new Logica(a, Operacion.NOT, aleft, aright); :}

         | PAR_A expresion:a PAR_C      {: RESULT = a; :}

         | ENTERO:a   {: RESULT = new Primitivo(Double.parseDouble(a), aleft, aright); :}
         | DECIMAL:a  {: RESULT = new Primitivo(Double.parseDouble(a), aleft, aright); :}
         | LIT_STRING:a   {: RESULT = new Primitivo(a, aleft, aright); :}
         | LIT_CHAR:a     {: RESULT = new Primitivo(a, aleft, aright); :}
         | TRUE:a         {: RESULT = new Primitivo(true, aleft, aright); :}
         | FALSE:a        {: RESULT = new Primitivo(false, aleft, aright); :}

         | ID:a         {: RESULT = new AccesoVar(a, aleft, aright); :};
```