```
package analizadores;

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

parser code
{:
   public LinkedList<Instruccion> AST;

   public void syntax_error(Symbol s){
      System.err.println("Error Sintáctico en la Línea " + s.left + " Columna " + s.right + ". No se
esperaba este componente: " +s.value);
      Errores.agregar("Sintáctico", "No se esperaba: " + s.value, s.left, s.right);
   }

   public void unrecovered_syntax_error(Symbol s) throws java.lang.Exception{
      System.err.println("Error sintáctico irrecuperable en la Línea " + s.left + " Columna " + s.right + ".
Componente " + s.value + " no reconocido.");
      Errores.agregar("Sintáctico Irrecuperable", "Componente no reconocido: " + s.value, s.left,
s.right);
   }
:}

/* --- TERMINALES --- */
terminal String INT, DOUBLE, BOOL, CHAR, STRING, VOID;
terminal VAR, IF, ELSE, SWITCH, CASE, DEFAULT;
terminal WHILE, DO, FOR;
terminal BREAK, CONTINUE, RETURN;
terminal START_WITH;
terminal PRINT, PRINTLN;
terminal LENGTH, ROUND, TOSTRING, FIND;
terminal NEW, LIST, APPEND, REMOVE;
terminal String TRUE, FALSE;
terminal MAS, MENOS, POR, DIV, POTENCIA, MOD;
terminal MASMAS, MENOSMENOS;
terminal IGUALIGUAL, DIFERENTE, MENOR, MENORIGUAL, MAYOR, MAYORIGUAL;
terminal AND, OR, NOT, XOR;
terminal PAR_A, PAR_C, LLAVE_A, LLAVE_C, COR_IZQ, COR_DER;
terminal PTCOMA, DOSPTOS, COMA, PUNTO;
terminal IGUAL;
terminal String ID, ENTERO, DECIMAL, LIT_STRING, LIT_CHAR;

/* --- NO TERMINALES --- */
non terminal LinkedList<Instruccion> ini;
non terminal LinkedList<Instruccion> instrucciones_globales;
non terminal Instruccion instruccion_global;
non terminal LinkedList<Instruccion> instrucciones;
non terminal Instruccion instruccion;
non terminal Instruccion declaracion, asignacion;
non terminal String tipo;
non terminal Instruccion funcion, parametro, llamada_funcion, sentencia_return, start_with;
non terminal LinkedList<Instruccion> lista_parametros, lista_argumentos;
non terminal Instruccion impresion, sentencia_if, sentencia_switch, caso, sentencia_while,
sentencia_do_while, sentencia_for, sentencia_break, sentencia_continue, actualizacion_for;
non terminal LinkedList<Instruccion> lista_casos;
```

```
non terminal Instruccion declaracion_vector, modificacion_vector, declaracion_lista, metodo_lista;
non terminal LinkedList<Instruccion> lista_valores;
non terminal Instruccion expresion;

/* --- 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 DIV, POR, MOD;
precedence right POTENCIA;
precedence right MASMAS, MENOSMENOS;

start with ini;

ini ::= instrucciones_globales:a {: AST = a; :};

instrucciones_globales ::= instrucciones_globales:a instruccion_global:b {: if(b!=null) a.add(b); RESULT
= a; :}
                    | instruccion_global:a {: LinkedList<Instruccion> l = new LinkedList<>(); if(a!=null)
l.add(a); RESULT = l; :};

/* ============ INSTRUCCIONES GLOBALES (SE EJECUTA UNA SOLA VEZ ESTA DEFINICIÓN)
============ */
instruccion_global ::= declaracion:a      {: RESULT = a; :}
                | declaracion_vector:a {: RESULT = a; :}
                | declaracion_lista:a {: RESULT = a; :}
                | funcion:a         {: RESULT = a; :}
                | start_with:a      {: RESULT = a; :}
                | asignacion:a      {: RESULT = a; :}
                | modificacion_vector:a {: RESULT = a; :}
                | impresion:a       {: RESULT = a; :}
                | sentencia_if:a    {: RESULT = a; :}
                | sentencia_switch:a {: RESULT = a; :}
                | sentencia_while:a  {: RESULT = a; :}
                | sentencia_do_while:a {: RESULT = a; :}
                | sentencia_for:a    {: RESULT = a; :}
                | sentencia_break:a  {: RESULT = a; :}
                | sentencia_continue:a {: RESULT = a; :}
                | llamada_funcion:a PTCOMA {: RESULT = a; :}
                | metodo_lista:a PTCOMA {: RESULT = a; :};

start_with ::= START_WITH ID:id PAR_A PAR_C PTCOMA {: RESULT = new StartWith(id, idleft,
idright); :};

funcion ::= tipo:t ID:id PAR_A lista_parametros:p PAR_C LLAVE_A instrucciones:bloque LLAVE_C
        {: RESULT = new Funcion(t, id, p, bloque, tleft, tright); :}
        | VOID ID:id PAR_A lista_parametros:p PAR_C LLAVE_A instrucciones:bloque LLAVE_C
        {: RESULT = new Funcion("void", id, p, bloque, idleft, idright); :};

lista_parametros ::= lista_parametros:l COMA parametro:p {: l.add(p); RESULT = l; :}
                | parametro:p {: LinkedList<Instruccion> l = new LinkedList<>(); l.add(p); RESULT = l; :}
                | /* vacio */ {: RESULT = new LinkedList<Instruccion>(); :};

parametro ::= tipo:t ID:id {: RESULT = new Declaracion(id, t, null, tleft, tright); :};
```

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

/* ============ INSTRUCCIONES DENTRO DE FUNCIONES/BLOQUES ============ */
instruccion ::= declaracion:a      {: RESULT = a; :}
          | declaracion_vector:a {: RESULT = a; :}
          | declaracion_lista:a {: RESULT = a; :}
          | asignacion:a       {: RESULT = a; :}
          | modificacion_vector:a {: RESULT = a; :}
          | impresion:a        {: RESULT = a; :}
          | sentencia_if:a     {: RESULT = a; :}
          | sentencia_switch:a  {: RESULT = a; :}
          | sentencia_while:a   {: RESULT = a; :}
          | sentencia_do_while:a {: RESULT = a; :}
          | sentencia_for:a     {: RESULT = a; :}
          | sentencia_break:a   {: RESULT = a; :}
          | sentencia_continue:a {: RESULT = a; :}
          | sentencia_return:a  {: RESULT = a; :}
          | llamada_funcion:a PTCOMA {: RESULT = a; :}
          | metodo_lista:a PTCOMA    {: RESULT = a; :};

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); :};

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

asignacion ::= ID:id IGUAL expresion:e PTCOMA
         {: RESULT = new Asignacion(id, e, idleft, idright); :}
         | ID:id MASMAS PTCOMA
         {: RESULT = new Asignacion(id, new Aritmetica(new AccesoVar(id, idleft, idright), new Dato("1",
"int", 0, 0), Operacion.Tipo_Operacion.SUMA, idleft, idright), idleft, idright); :}
         | ID:id MENOSMENOS PTCOMA
         {: RESULT = new Asignacion(id, new Aritmetica(new AccesoVar(id, idleft, idright), new Dato("1",
"int", 0, 0), Operacion.Tipo_Operacion.RESTA, idleft, idright), idleft, idright); :};

declaracion_vector ::= VAR ID:id DOSPTOS tipo:t COR_IZQ COR_DER IGUAL COR_IZQ lista_valores:l
COR_DER PTCOMA
               {: RESULT = new DeclaracionVector(id, t, 1, l, idleft, idright); :}
               | VAR ID:id DOSPTOS tipo:t COR_IZQ COR_DER COR_IZQ COR_DER IGUAL COR_IZQ
lista_valores:l COR_DER PTCOMA
               {: RESULT = new DeclaracionVector(id, t, 2, l, idleft, idright); :};

lista_valores ::= lista_valores:l COMA expresion:e {: l.add(e); RESULT = l; :}
          | expresion:e {: LinkedList<Instruccion> l = new LinkedList<>(); l.add(e); RESULT = l; :};

modificacion_vector ::= ID:id COR_IZQ expresion:ind COR_DER IGUAL expresion:val PTCOMA
               {: RESULT = new ModificacionVector(id, ind, null, val, idleft, idright); :}
               | ID:id COR_IZQ expresion:i COR_DER COR_IZQ expresion:j COR_DER IGUAL expresion:val
PTCOMA
```

```
                {: RESULT = new ModificacionVector(id, i, j, val, idleft, idright); :};

declaracion_lista ::= LIST MENOR tipo:t MAYOR ID:id IGUAL NEW LIST PAR_A PAR_C PTCOMA
                {: RESULT = new DeclaracionLista(id, t, idleft, idright); :};

metodo_lista ::= ID:id PUNTO APPEND PAR_A expresion:e PAR_C
            {: RESULT = new MetodoLista(id, "append", e, idleft, idright); :}
            | ID:id PUNTO REMOVE PAR_A expresion:e PAR_C
            {: RESULT = new MetodoLista(id, "remove", e, idleft, idright); :};

impresion ::= PRINT PAR_A expresion:e PAR_C PTCOMA
          {: RESULT = new Print(e, false, eleft, eright); :}
          | PRINTLN PAR_A expresion:e PAR_C PTCOMA
          {: RESULT = new Print(e, true, eleft, eright); :};

sentencia_if ::= IF PAR_A expresion:cond PAR_C LLAVE_A instrucciones:i1 LLAVE_C
            {: RESULT = new If(cond, i1, null, condleft, condright); :}
            | IF PAR_A expresion:cond PAR_C LLAVE_A instrucciones:i1 LLAVE_C ELSE LLAVE_A
instrucciones:i2 LLAVE_C
            {: RESULT = new If(cond, i1, i2, condleft, condright); :}
            | IF PAR_A expresion:cond PAR_C LLAVE_A instrucciones:i1 LLAVE_C ELSE sentencia_if:i2
            {: LinkedList<Instruccion> l = new LinkedList<>(); l.add(i2); RESULT = new If(cond, i1, l,
condleft, condright); :};

sentencia_switch ::= SWITCH PAR_A expresion:e PAR_C LLAVE_A lista_casos:l LLAVE_C
                {: RESULT = new Switch(e, l, eleft, eright); :};

lista_casos ::= lista_casos:l caso:c {: if(c!=null) l.add(c); RESULT = l; :}
            | caso:c {: LinkedList<Instruccion> l = new LinkedList<>(); if(c!=null) l.add(c); RESULT = l; :};

caso ::= CASE expresion:e DOSPTOS instrucciones:i {: RESULT = new Caso(e, i, eleft, eright); :}
      | DEFAULT DOSPTOS instrucciones:i {: RESULT = new Caso(null, i, ileft, iright); :};

sentencia_while ::= WHILE PAR_A expresion:cond PAR_C LLAVE_A instrucciones:i LLAVE_C
                {: RESULT = new While(cond, i, condleft, condright); :};

sentencia_do_while ::= DO LLAVE_A instrucciones:i LLAVE_C WHILE PAR_A expresion:cond PAR_C
PTCOMA
                {: RESULT = new DoWhile(cond, i, condleft, condright); :};

actualizacion_for ::= ID:id IGUAL expresion:e {: RESULT = new Asignacion(id, e, idleft, idright); :}
                | ID:id MASMAS
                {: RESULT = new Asignacion(id, new Aritmetica(new AccesoVar(id,0,0), new
Dato("1","int",0,0), Operacion.Tipo_Operacion.SUMA, 0,0), idleft, idright); :}
                | ID:id MENOSMENOS
                {: RESULT = new Asignacion(id, new Aritmetica(new AccesoVar(id,0,0), new
Dato("1","int",0,0), Operacion.Tipo_Operacion.RESTA, 0,0), idleft, idright); :};

sentencia_for ::= FOR PAR_A asignacion:ini expresion:cond PTCOMA actualizacion_for:inc PAR_C
LLAVE_A instrucciones:i LLAVE_C
            {: RESULT = new For(ini, cond, inc, i, inileft, iniright); :};

sentencia_break ::= BREAK:b PTCOMA {: RESULT = new Break(bleft, bright); :};
sentencia_continue ::= CONTINUE:c PTCOMA {: RESULT = new Continue(cleft, cright); :};

sentencia_return ::= RETURN expresion:e PTCOMA {: RESULT = new Return(e, eleft, eright); :}
                | RETURN PTCOMA {: RESULT = new Return(null, 0, 0); :};
```

```
expresion ::= MENOS expresion:a {: RESULT = new Aritmetica(a,
Operacion.Tipo_Operacion.NEGACION, aleft, aright); :} %prec POTENCIA
        | PAR_A INT PAR_C expresion:e {: RESULT = new Casteo("int", e, eleft, eright); :} %prec
POTENCIA
        | PAR_A DOUBLE PAR_C expresion:e {: RESULT = new Casteo("double", e, eleft,
eright); :} %prec POTENCIA
        | PAR_A CHAR PAR_C expresion:e {: RESULT = new Casteo("char", e, eleft, eright); :} %prec
POTENCIA
        | PAR_A STRING PAR_C expresion:e {: RESULT = new Casteo("string", e, eleft, eright); :} %prec
POTENCIA
        | PAR_A BOOL PAR_C expresion:e {: RESULT = new Casteo("bool", e, eleft, eright); :} %prec
POTENCIA
        | expresion:a MAS expresion:b {: RESULT = new Aritmetica(a, b,
Operacion.Tipo_Operacion.SUMA, aleft, aright); :}
        | expresion:a MENOS expresion:b {: RESULT = new Aritmetica(a, b,
Operacion.Tipo_Operacion.RESTA, aleft, aright); :}
        | expresion:a POR expresion:b {: RESULT = new Aritmetica(a, b,
Operacion.Tipo_Operacion.MULTIPLICACION, aleft, aright); :}
        | expresion:a DIV expresion:b {: RESULT = new Aritmetica(a, b,
Operacion.Tipo_Operacion.DIVISION, aleft, aright); :}
        | expresion:a POTENCIA expresion:b {: RESULT = new Aritmetica(a, b,
Operacion.Tipo_Operacion.POTENCIA, aleft, aright); :}
        | expresion:a MOD expresion:b {: RESULT = new Aritmetica(a, b,
Operacion.Tipo_Operacion.MODULO, aleft, aright); :}
        | expresion:a IGUALIGUAL expresion:b {: RESULT = new Logica(a, b,
Operacion.Tipo_Operacion.IGUALIGUAL, aleft, aright); :}
        | expresion:a DIFERENTE expresion:b {: RESULT = new Logica(a, b,
Operacion.Tipo_Operacion.DIFERENTE, aleft, aright); :}
        | expresion:a MENOR expresion:b {: RESULT = new Logica(a, b,
Operacion.Tipo_Operacion.MENOR, aleft, aright); :}
        | expresion:a MAYOR expresion:b {: RESULT = new Logica(a, b,
Operacion.Tipo_Operacion.MAYOR, aleft, aright); :}
        | expresion:a MENORIGUAL expresion:b {: RESULT = new Logica(a, b,
Operacion.Tipo_Operacion.MENORIGUAL, aleft, aright); :}
        | expresion:a MAYORIGUAL expresion:b {: RESULT = new Logica(a, b,
Operacion.Tipo_Operacion.MAYORIGUAL, aleft, aright); :}
        | expresion:a AND expresion:b {: RESULT = new Logica(a, b, Operacion.Tipo_Operacion.AND,
aleft, aright); :}
        | expresion:a OR expresion:b {: RESULT = new Logica(a, b, Operacion.Tipo_Operacion.OR, aleft,
aright); :}
        | expresion:a XOR expresion:b {: RESULT = new Logica(a, b, Operacion.Tipo_Operacion.XOR,
aleft, aright); :}
        | NOT expresion:a {: RESULT = new Logica(a, Operacion.Tipo_Operacion.NOT, aleft, aright); :}
        | PAR_A expresion:e PAR_C {: RESULT = e; :}
        | ID:id {: RESULT = new AccesoVar(id, idleft, idright); :}
        | ENTERO:e {: RESULT = new Dato(e, "int", eleft, eright); :}
        | DECIMAL:e {: RESULT = new Dato(e, "double", eleft, eright); :}
        | LIT_STRING:e {: RESULT = new Dato(e, "string", eleft, eright); :}
        | LIT_CHAR:e {: RESULT = new Dato(e, "char", eleft, eright); :}
        | TRUE {: RESULT = new Dato("true", "bool", 0, 0); :}
        | FALSE {: RESULT = new Dato("false", "bool", 0, 0); :}
        | COR_IZQ lista_valores:l COR_DER {: RESULT = new VectorLiteral(l, lleft, lright); :}
        | ID:id COR_IZQ expresion:i COR_DER {: RESULT = new AccesoVector(id, i, null, idleft, idright); :}
        | ID:id COR_IZQ expresion:i COR_DER COR_IZQ expresion:j COR_DER {: RESULT = new
AccesoVector(id, i, j, idleft, idright); :}
        | ROUND PAR_A expresion:e PAR_C {: RESULT = new Nativa(e, "round", eleft, eright); :}
```

```
        | LENGTH PAR_A expresion:e PAR_C {: RESULT = new Nativa(e, "length", eleft, eright); :}
        | TOSTRING PAR_A expresion:e PAR_C {: RESULT = new Nativa(e, "toString", eleft, eright); :}
        | ID:id PUNTO FIND PAR_A expresion:e PAR_C {: RESULT = new FindVector(id, e, idleft,
idright); :}
        | ID:id PUNTO REMOVE PAR_A expresion:e PAR_C {: RESULT = new MetodoLista(id, "remove",
e, idleft, idright); :}
        | llamada_funcion:l {: RESULT = l; :};

llamada_funcion ::= ID:id PAR_A lista_argumentos:l PAR_C {: RESULT = new LlamadaFuncion(id, l,
idleft, idright); :};

lista_argumentos ::= lista_argumentos:l COMA expresion:e {: l.add(e); RESULT = l; :}
            | expresion:e {: LinkedList<Instruccion> l = new LinkedList<>(); l.add(e); RESULT = l; :}
            | /* vacio */ {: RESULT = new LinkedList<Instruccion>(); :};
```