Terminales:

terminal VAR, INT, DOUBLE, BOOL, CHAR, STRINGTYPE;

terminal IF, ELSE;

terminal WHILE, FOR, DO;

terminal BREAK, CONTINUE;

terminal PRINT;

terminal String CADENA, ENTERO, DECIMAL;

terminal char CARACTER;

terminal boolean BOOLEANO;

terminal String IDENTIFICADOR;

terminal MAS, MENOS, MULT, DIV, POTENCIA, MODULO;

terminal INCREMENTO, DECREMENTO;

terminal IGUAL, DIFERENTE, MENOR, MENOR_IGUAL, MAYOR, MAYOR_IGUAL;

terminal AND, OR, XOR, NOT;

terminal PAR_ABRIR, PAR_CERRAR;

terminal LLAVE_ABRIR, LLAVE_CERRAR;

terminal PUNTO_COMA, DOS_PUNTOS;

terminal SWITCH, CASE, DEFAULT;

terminal ASIG;

terminal UMINUS;

No terminales:

nonterminal LinkedList INICIO, INSTRUCCIONES, LISTA_CASES, ELSEIF_LIST, DEFAULT_INST;

nonterminal Instruccion INSTRUCCION, EXPRESION, DECLARACION, ASIGNACION, SIF, WHILE_LOOP, DO_WHILE_LOOP, FOR_LOOP, BREAK_INST, CONTINUE_INST, SWITCH_INST;

nonterminal SwitchCaso CASE_INST;

nonterminal Tipo TIPO;

Precedencias:

precedence left OR;

precedence left AND;

precedence left XOR;

precedence right NOT;

precedence left IGUAL, DIFERENTE, MENOR, MAYOR, MENOR_IGUAL, MAYOR_IGUAL;

precedence left MAS, MENOS;

precedence left MULT, DIV, MODULO;

precedence right POTENCIA;

precedence right UMINUS;

Siempre iniciamos con INICIO:

start with INICIO;

Las reglas que usaremos:

INICIO ::= INSTRUCCIONES:a {: RESULT = a; :};

INSTRUCCIONES ::= INSTRUCCIONES:a INSTRUCCION:b {: RESULT = a; a.add(b); :}

| INSTRUCCION:a {: RESULT = new LinkedList(); RESULT.add(a); :};

INSTRUCCION ::= PRINT PAR_ABRIR EXPRESION:a PAR_CERRAR PUNTO_COMA {: RESULT = new Print(a, aleft, aright); :}

| DECLARACION:a PUNTO_COMA {: RESULT = a; :}

| DECLARACION:a {: RESULT = a; :}

| ASIGNACION:a PUNTO_COMA {: RESULT = a; :}

| ASIGNACION:a {: RESULT = a; :}

| SIF:a {: RESULT = a; :}

| WHILE_LOOP:a {: RESULT = a; :}

| DO_WHILE_LOOP:a {: RESULT = a; :}

| FOR_LOOP:a {: RESULT = a; :}

| SWITCH_INST:a {: RESULT = a; :}

| BREAK_INST:a {: RESULT = a; :}

| CONTINUE_INST:a {: RESULT = a; :}

```
        | IDENTIFICADOR:a INCREMENTO {: RESULT = new Incremento(a, true, aleft, aright); :}

        | IDENTIFICADOR:a DECREMENTO {: RESULT = new Incremento(a, false, aleft, aright); :}

        | IDENTIFICADOR:a INCREMENTO PUNTO_COMA {: RESULT = new Incremento(a, true,
aleft, aright); :}

        | IDENTIFICADOR:a DECREMENTO PUNTO_COMA {: RESULT = new Incremento(a, false,
aleft, aright); :};


WHILE_LOOP ::= WHILE PAR_ABRIR EXPRESION:a PAR_CERRAR LLAVE_ABRIR
INSTRUCCIONES:b LLAVE_CERRAR

        {: RESULT = new While(a, b, aleft, aright); :};


DO_WHILE_LOOP ::= DO LLAVE_ABRIR INSTRUCCIONES:a LLAVE_CERRAR WHILE PAR_ABRIR
EXPRESION:b PAR_CERRAR PUNTO_COMA

          {: RESULT = new Do_While(b, a, new Tipo(Datos.VOID), bleft, bright); :};


FOR_LOOP ::=

  FOR PAR_ABRIR DECLARACION:a PUNTO_COMA EXPRESION:b PUNTO_COMA INSTRUCCION:c
PAR_CERRAR

   LLAVE_ABRIR INSTRUCCIONES:d LLAVE_CERRAR

   {: RESULT = new For(a, b, c, d, new Tipo(Datos.VOID), aleft, aright); :}

| FOR PAR_ABRIR ASIGNACION:a PUNTO_COMA EXPRESION:b PUNTO_COMA INSTRUCCION:c
PAR_CERRAR

   LLAVE_ABRIR INSTRUCCIONES:d LLAVE_CERRAR

   {: RESULT = new For(a, b, c, d, new Tipo(Datos.VOID), aleft, aright); :};


SIF ::= IF PAR_ABRIR EXPRESION:a PAR_CERRAR LLAVE_ABRIR INSTRUCCIONES:b
LLAVE_CERRAR

      {: RESULT = new If(a, b, aleft, aright); :}

    | IF PAR_ABRIR EXPRESION:a PAR_CERRAR LLAVE_ABRIR INSTRUCCIONES:b LLAVE_CERRAR
ELSE LLAVE_ABRIR INSTRUCCIONES:c LLAVE_CERRAR

      {: RESULT = new If(a, b, c, aleft, aright); :}

    | IF PAR_ABRIR EXPRESION:a PAR_CERRAR LLAVE_ABRIR INSTRUCCIONES:b LLAVE_CERRAR
ELSEIF_LIST:d

      {: RESULT = new If(a, b, d, null, new Tipo(Datos.VOID), aleft, aright); :}
```

```
    | IF PAR_ABRIR EXPRESION:a PAR_CERRAR LLAVE_ABRIR INSTRUCCIONES:b LLAVE_CERRAR
ELSEIF_LIST:d ELSE LLAVE_ABRIR INSTRUCCIONES:e LLAVE_CERRAR

      {: RESULT = new If(a, b, d, e, new Tipo(Datos.VOID), aleft, aright); :};


ELSEIF_LIST ::= ELSEIF_LIST:a ELSE IF PAR_ABRIR EXPRESION:b PAR_CERRAR LLAVE_ABRIR
INSTRUCCIONES:c LLAVE_CERRAR

           {: RESULT = a; a.add(new If(b, c, bleft, bright)); :}

          | ELSE IF PAR_ABRIR EXPRESION:a PAR_CERRAR LLAVE_ABRIR INSTRUCCIONES:b
LLAVE_CERRAR

           {: RESULT = new LinkedList(); RESULT.add(new If(a, b, aleft, aright)); :};


BREAK_INST ::= BREAK:b PUNTO_COMA {: RESULT = new Break(bleft, bright); :};


CONTINUE_INST ::= CONTINUE:c PUNTO_COMA {: RESULT = new Continue(cleft, cright); :};


DECLARACION ::= VAR IDENTIFICADOR:a DOS_PUNTOS TIPO:b ASIG EXPRESION:c

           {: RESULT = new Declaracion(a, c, b, aleft, aright); :}

          | VAR IDENTIFICADOR:a DOS_PUNTOS TIPO:b

           {: RESULT = new Declaracion(a, null, b, aleft, aright); :};


LISTA_CASES ::= LISTA_CASES:a CASE_INST:b {: RESULT = a; a.add(b); :}

          | CASE_INST:a {: RESULT = new LinkedList(); RESULT.add(a); :};


CASE_INST ::= CASE EXPRESION:a DOS_PUNTOS INSTRUCCIONES:b

          {: RESULT = new SwitchCaso(a, b); :};


DEFAULT_INST ::= DEFAULT DOS_PUNTOS INSTRUCCIONES:a {: RESULT = a; :}

           | {: RESULT = null; :};


SWITCH_INST ::= SWITCH PAR_ABRIR EXPRESION:a PAR_CERRAR LLAVE_ABRIR LISTA_CASES:b
DEFAULT_INST:c LLAVE_CERRAR

           {: RESULT = new Switch(a, extraerValores(b), extraerInstrucciones(b), c, new
Tipo(Datos.VOID), aleft, aright); :}
```

```
        | SWITCH PAR_ABRIR EXPRESION:a PAR_CERRAR LLAVE_ABRIR DEFAULT_INST:c
LLAVE_CERRAR

        {: RESULT = new Switch(a, new LinkedList(), new LinkedList(), c, new
Tipo(Datos.VOID), aleft, aright); :};


ASIGNACION ::= IDENTIFICADOR:a ASIG EXPRESION:b

        {: RESULT = new Asignacion(a, b, aleft, aright); :};


Los tipos de las variables:

TIPO ::= INT {: RESULT = new Tipo(Datos.ENTERO); :}
    | DOUBLE {: RESULT = new Tipo(Datos.DECIMAL); :}
    | BOOL {: RESULT = new Tipo(Datos.BOOLEANO); :}
    | CHAR {: RESULT = new Tipo(Datos.CARACTER); :}
    | STRINGTYPE {: RESULT = new Tipo(Datos.CADENA); :};


Las expresiones que usaremos:

EXPRESION ::= EXPRESION:a OR EXPRESION:b
        {: RESULT = new Logicas(a, b, OperadoresLogicos.OR, aleft, aright); :}
        | EXPRESION:a AND EXPRESION:b
        {: RESULT = new Logicas(a, b, OperadoresLogicos.AND, aleft, aright); :}
        | EXPRESION:a XOR EXPRESION:b
        {: RESULT = new Logicas(a, b, OperadoresLogicos.XOR, aleft, aright); :}
        | NOT EXPRESION:a
        {: RESULT = new Logicas(null, a, OperadoresLogicos.NOT, aleft, aright); :}
        | EXPRESION:a IGUAL EXPRESION:b
        {: RESULT = new Relacionales(a, b, OperadoresRelacionales.IGUAL, aleft, aright); :}
        | EXPRESION:a DIFERENTE EXPRESION:b
        {: RESULT = new Relacionales(a, b, OperadoresRelacionales.DIFERENTE, aleft, aright); :}
        | EXPRESION:a MENOR EXPRESION:b
        {: RESULT = new Relacionales(a, b, OperadoresRelacionales.MENOR, aleft, aright); :}
        | EXPRESION:a MAYOR EXPRESION:b
```

```
        {: RESULT = new Relacionales(a, b, OperadoresRelacionales.MAYOR, aleft, aright); :}
      | EXPRESION:a MENOR_IGUAL EXPRESION:b
        {: RESULT = new Relacionales(a, b, OperadoresRelacionales.MENOR_IGUAL, aleft,
aright); :}
      | EXPRESION:a MAYOR_IGUAL EXPRESION:b
        {: RESULT = new Relacionales(a, b, OperadoresRelacionales.MAYOR_IGUAL, aleft,
aright); :}
      | EXPRESION:a MAS EXPRESION:b
        {: RESULT = new Aritmetica(a, b, OperadoresAritmeticos.SUMA, aleft, aright); :}
      | EXPRESION:a MENOS EXPRESION:b
        {: RESULT = new Aritmetica(a, b, OperadoresAritmeticos.RESTA, aleft, aright); :}
      | EXPRESION:a MULT EXPRESION:b
        {: RESULT = new Aritmetica(a, b, OperadoresAritmeticos.MULTIPLICACION, aleft,
aright); :}
      | EXPRESION:a DIV EXPRESION:b
        {: RESULT = new Aritmetica(a, b, OperadoresAritmeticos.DIVISION, aleft, aright); :}
      | EXPRESION:a MODULO EXPRESION:b
        {: RESULT = new Aritmetica(a, b, OperadoresAritmeticos.MODULO, aleft, aright); :}
      | EXPRESION:a POTENCIA EXPRESION:b
        {: RESULT = new Aritmetica(a, b, OperadoresAritmeticos.POTENCIA, aleft, aright); :}
      | MENOS EXPRESION:a
        {: RESULT = new Aritmetica(OperadoresAritmeticos.NEGACION, a, aleft, aright); :}
%prec UMINUS
      | PAR_ABRIR TIPO:a PAR_CERRAR EXPRESION:b
        {: RESULT = new Casteo(a.getTipo(), b, aleft, aright); :}
      | PAR_ABRIR EXPRESION:a PAR_CERRAR
        {: RESULT = a; :}
      | ENTERO:a
        {: RESULT = new Nativo(Integer.valueOf(a), new Tipo(Datos.ENTERO), aleft, aright); :}
      | DECIMAL:a
        {: RESULT = new Nativo(Double.valueOf(a), new Tipo(Datos.DECIMAL), aleft, aright); :}
      | CARACTER:a
```

```
 {: RESULT = new Nativo(a, new Tipo(Datos.CARACTER), aleft, aright); :}
| CADENA:a
 {: RESULT = new Nativo(a, new Tipo(Datos.CADENA), aleft, aright); :}
| BOOLEANO:a
 {: RESULT = new Nativo(a, new Tipo(Datos.BOOLEANO), aleft, aright); :}
| IDENTIFICADOR:a
 {: RESULT = new AccesoVar(a, aleft, aright); :}
;
```