PROCESADORES DE LENGUAJES

FASE 3.2:

DESARROLLO DE CONSTRUCTORES DE ASTs PARA TINY(1).

Grupo 07:
HongXiang Chen y
Andrés Teruel Fernández

1. Especificación de la sintaxis abstracta.

Constructora	Explicación
prog: Decs x Instrs → Prog	Construye un programa, dados una lista de declaraciones y una lista de instrucciones.
decs_muchas: Decs x Dec → Decs	Dada una lista de declaraciones y una declaracion, construye una lista de declaraciones.
decs_una: Dec → Decs	Dada una declaracion, construye una lista de declaraciones.
decs_vacia: → Decs	Construye una lista de declaraciones
dec_var: Tipo x string → Dec	Dado su tipo y su indentificador, construye una declaracion de variables.
dec_tipo: Tipo x string → Dec	Dado su tipo y su indentificador, construye una declaracion de tipo.
dec_proc: string x ParamsForm x Bloque→ Dec	Dado su bloque, sus parámetros formales y su indentificador, construye una declaracion de procedimiento.
paramForm_muchos: Params x Param → ParamsForm	Dado dos parámetros, construye una lista de parámetros formales.
paramForm_uno: Param→ ParamsForm	Dado un parámetros, construye una lista de parámetros formales.
paramForm_vacio: ParamsForm	Construye una lista de parámetros formales.
param_sin_amp : Tipo x string → Param	Dado su tipo y su indentificador, construye un parámetro.
param_con_amp : Tipo x string → Param	Dado su tipo y su indentificador, construye un parámetro con ampersand.
bloque: Decs x Instrs → Bloque	Dada una lista de declaraciones y una lista de instrucciones, construye un bloque
tipoInt: Tipo	Construye el tipo int.
tipoReal: Tipo	Construye el tipo real.
tipoBool: Tipo	Construye el tipo bool.
tipoString: Tipo	Construye el tipo string.

tipoId: string → Tipo	Construye un tipo personalizado con el valor de string.
tipoArray: Num_ent x Tipo → Tipo	Construye el tipo Array.
tipoRegistro: Campos → Tipo	Dada una lista de campo, contruye un tipo registro
tipoPuntero: Tipo → Tipo	Construye un tipo puntero
campos_muchos: Campos x Campo → Campos	Dada una lista de campo y un campo, construye una lista de campo.
campos_uno: Campo → Campos	Dado un campo, construye una lista de campo.
campo: Tipo x string → Campo	Dado un tipo y un string, construye un campo
instr_muchas: Instrs x Instr → Instrs	Dada una lista de instrucciones y una instrucción, construye una lista de instrucciones.
instr_una : Instr → Instrs	Dada una instrucción, construye una lista de instrucciones .
instr_vacia: Instrs	Construye una lista de instrucciones
instruccionAsig: Exp x Exp → Instr	Dados dos expresión, construye una instrucción de asignación
instruccionIf: Exp x Instrs → Instr	Dada una expresión y una lista de instrucciones, construye una instrucción de if.
instruccionIfElse: Exp x Instrs x Instrs → Instr	Dada una expresión y dos listas de instrucciones, construye una instrucción de if else.
instruccionWhile: Exp x Instrs → Instr	Dada una expresión y una lista de instrucciones, construye una instrucción de while.
instruccionRead: Exp → Instr	Dada una expresión, construye una instrucción de read.
instruccionWrite: Exp → Instr	Dada una expresión, construye una instrucción de write.
instruccionNew: Exp → Instr	Dada una expresión, construye una instrucción de new.
instruccionDelete: Exp → Instr	Dada una expresión, construye una instrucción de delete.
instruccionCall: string x ParamReales → Instr	Dada un string y una lista de parámetro reales, construye una instrucción call.

instruccionBloque: Bloque → Instr	Dada un bloque, construye una instrucción bloque
instruccionNl : Instr	Construye una instrucción Nl
paramReales_muchos: ParamReales x Exp → ParamReales	Dada una lista de parámetro reales y una lista de parámetro reales, construye una lista de parámetro reales.
paramReales_uno: Exp → ParamReales	Dada una expresión, construye una lista de parámetro reales.
paramReales_vacios: ParamReales	Construye una lista de parámetro reales.
num_ent: string → Exp	Construye una expresión que representa un entero a partir de la cadena asociada a dicho literal
num_real: string → Exp	Construye una expresión que representa un real a partir de la cadena asociada a dicho literal
id: string → Exp	Construye una expresión que representa una variable a partir del nombre de dicha variable
boolean: string → Exp	Construye una expresión que representa un booleano
cadena: string → Exp	Dada un string, construye una expresión
null: Exp	Construye una expresión
mas: Exp x Exp → Exp	Construye una expresión a partir de la suma de dos expresiones
menos: Exp x Exp → Exp	Construye una expresión a partir de la resta de dos expresiones
and: Exp x Exp → Exp	Construye una expresión a partir de lógica and de dos expresiones
or: Exp x Exp → Exp	Construye una expresión a partir de lógica or de dos expresiones
menor: Exp x Exp → Exp	Construye una expresión a partir de la comparación menor de dos expresiones
mayor: Exp x Exp → Exp	Construye una expresión a partir de la comparación mayor de dos expresiones
menor_igual: Exp x Exp → Exp	Construye una expresión a partir de la comparación menor igual de dos expresiones
mayor_igual: Exp x Exp → Exp	Construye una expresión a partir de la comparación mayor igual de dos expresiones

diferente: Exp x Exp → Exp	Construye una expresión a partir de la comparación de diferencia de dos expresiones
igual: Exp x Exp → Exp	Construye una expresión a partir de la comparación de igualdad de dos expresiones
mult: Exp x Exp → Exp	Construye una expresión a partir de la multiplicación de dos expresiones
asterisco: Exp → Exp	Construye una expresión a partir de una expresión asterisco
div: Exp x Exp → Exp	Construye una expresión a partir de la división de dos expresiones
porcentaje: Exp x Exp → Exp	Construye una expresión a partir del porcentaje de dos expresiones
not: Exp → Exp	Construye una expresión a partir de lógica not de una expresiones
negativo: Exp → Exp	Construye una expresión a partir del operador de negativo de una expresiones
indexacion: Exp x Exp → Exp	Construye una expresión a partir de la indexación de dos expresiones
acceso_registro_punto: Exp x string → Exp	Construye una expresión a partir de una expresión y un string.
acceso_registro_flecha: Exp x string → Exp	Construye una expresión a partir de una expresión y un string.

2. Especificación del constructor AST.

G = (Vn , Vt, P, S)

VN = {S, SecDec, Declaraciones, Declaracion, DeclaracionVar, DeclaracionTipo, DeclaracionProc, ParametrosFormales, ParametrosFormalesAux, Parametro, Bloque, Tipo, Array, Registro, Campos, Campo, Puntero, SecInstr, Instruccion, InstruccionCall, InstruccionAsignacion, InstruccionIf, SecInstr2, InstruccionWhile, InstruccionRead, InstruccionWrite, InstruccionNew, InstruccionDelete, ParametrosReales, ParametrosRealesAux, ExpresionBasica, E0, E1, E2, E3, E4, E5, E6, E7, Op1, Op2, Op3, Op4,}

VT = {NL, OF, TYPE, PROC, SEPARADOR, PUNTOCOMA, VAR, IDENTIFICADOR, PAR_ABIERTO, PAR_CERRADO, COMA, AMPERSAND, LLAVE_ABIERTA, LLAVE_CERRADA, INT, REAL, BOOL, STRING, ARRAY, RECORD, POINTER, CORCHETE_ABIERTO, CORCHETE_CERRADO, ASIGNACION, MAS, MENOS, NOT, NUM_ENT, NUM_REAL, BOOLEAN, AND, OR, MENOR, MAYOR, MENOR_IGUAL,

MAYOR_IGUAL, DIFERENTE, IGUAL, MULT, DIV, NULL, IF, ELSE, WHILE, ENDIF, ENDWHILE, DO, THEN, READ, WRITE, NEW, DELETE, CALL, PORCENTAJE, PUNTO, FLECHA , CADENA}

P = {

SAux → S |-;

SAux.a = S.a

S → SecDec SecInstr;

S.a = prog (SecDec.a, SecInstr.a).

SecDec → Declaraciones SEPARADOR;

SecDec.a = Declaraciones.a.

SecDec → ε;

SecDes.a = decs_vacia().

Declaraciones → Declaraciones PUNTOCOMA Declaracion;

Declaraciones0.a = decs_muchas(Declaraciones1.a, Declaracion.a);

Declaraciones → Declaracion;

Declaracións.a = decs_una(Declaracion.a).

Declaracion → DeclaracionVar;

Declaracion.a = DeclaracionVar.a.

Declaracion → DeclaracionTipo;

Declaracion.a = DeclaracionTipo.a.

Declaracion → DeclaracionProc;

Declaracion.a = DeclaracionProc.a.

DeclaracionVar → VAR Tipo IDENTIFICADOR;

DeclaracionVar.a = dec_var(Tipo.a, IDENTIFICADOR.lex).

DeclaracionTipo → TYPE Tipo IDENTIFICADOR;

DeclaracionTipo.a = dec_tipo(Tipo.a, IDENTIFICADOR.lex).

DeclaracionProc → PROC IDENTIFICADOR PAR_ABIERTO ParametrosFormales PAR_CERRADO Bloque;

DeclaracionProc.a = dec_proc(IDENTIFICADOR.lex, ParametrosFormales.a, Bloque.a).

ParametrosFormales → ParametrosFormalesAux;

ParametrosFormales.a = ParametrosFormalesAux.a.

ParametrosFormales → ε;

ParametrosFormales.a = paramForm_vacio().

ParametrosFormalesAux → ParametrosFormalesAux COMA Parametro;

ParametrosFormalesAux.a = paramForm_muchos(ParametrosFormalesAux.a, Parametro.a).

ParametrosFormalesAux → Parametro;

ParametrosFormalesAux.a = paramForm_uno(Parametro.a).

Parametro → Tipo IDENTIFICADOR;

parametro.a = param_sin_amp(Tipo.a, IDENTIFICADOR.lex).

Parametro → Tipo AMPERSAND IDENTIFICADOR;

parametro.a = param_con_amp(Tipo.a, IDENTIFICADOR.lex).

Bloque → LLAVE_ABIERTA SecDec SecInstr LLAVE_CERRADA;

Bloque.a = bloque(SecDec.a, SecInstr.a).

Tipo → INT ;

Tipo.a = tipoInt().

Tipo → REAL ;

Tipo.a = tipoRealt().

Tipo → BOOL ;

Tipo.a = tipoBool().

Tipo → STRING ;

Tipo.a = tipoString().

Tipo → IDENTIFICADOR ;

Tipo.a = tipoId(IDENTIFICADOR.lex).

Tipo → Array;

Tipo.a = array.a.

Tipo → Registro ;

Tipo.a = resgistro.a.

Tipo → Puntero;

Tipo.a = puntero.a.

Array → ARRAY CORCHETE_ABIERTO NUM_ENT CORCHETE_CERRADO OF Tipo;

Array.a = tipoArray(NUN_ENT.lex, Tipo.a).

Registro → RECORD LLAVE_ABIERTA Campos LLAVE_CERRADA;

Registro.a = tipoRegistro(Campos.a).

Campos → Campos PUNTOCOMA Campo;

Campos0.a = campos_muchos(Campos1.a, Campo.a).

Campos → Campo;

Campos.a = campos_uno(Campo.a).

Campo → Tipo IDENTIFICADOR;

Campo.a = campo(Tipo.a, IDENTIFICADOR.lex).

Puntero → POINTER Tipo;

Puntero.a = tipoPuntero(Tipo.a).

SecInstr → SecInstr PUNTOCOMA Instruccion;

SecInstr.a = instr_muchas(SecInstr.a, Instruccion.a).

SecInstr → Instruccion;

SecInstr.a = instr_una(Instruccion.a).

Instruccion → InstruccionAsignacion;

Instruccion.a = InstruccionAsig.a.

Instruccion → InstruccionIf ;

Instruccion.a = InstruccionIF.a.

Instruccion → InstruccionWhile ;

Instruccion.a = InstruccionWhile.a.

Instruccion → InstruccionRead ;

Instruccion.a = InstruccionRead.a.

Instruccion → InstruccionWrite ;

Instruccion.a = InstruccionWrite.a.

Instruccion → NL ;

Instruccion.a = InstruccionNL().

Instruccion → InstruccionNew ;

Instruccion.a = InstruccionNew.a.

Instruccion → InstruccionDelete ;

Instruccion.a = InstruccionDelete.a.

Instruccion → InstruccionCall ;

Instruccion.a = InstruccionCall.a.

Instruccion → Bloque;

Instruccion.a = instruccionBloque(Bloque.a).

InstruccionAsignacion → E0 ASIGNACION E0;

InstruccionAsignacion.a = instruccionAsig(E0.a, E0.a).

InstruccionIf → IF E0 THEN SecInstr2 ENDIF;

InstruccionIf.a = instruccionIf(E0.a, SecInstr2.a).

InstruccionIf → IF E0 THEN SecInstr2 ELSE SecInstr2 ENDIF;

InstruccionIf.a = instruccionIfElse(E0.a, SecInstr21.a, SecInstr22.a).

SecInstr2 → SecInstr;

SecInstr2.a = SecInstr.a.

SecInstr2 → ε;

SecInstr2.a = instr_vacia().

InstruccionWhile → WHILE E0 DO SecInstr2 ENDWHILE;

InstruccionWhile.a = instruccionWhile(E0.a, SecInstr2.a).

InstruccionRead → READ E0;

InstruccionRead.a = instruccionRead(E0.a).

InstruccionWrite → WRITE E0;

InstruccionWrite.a = instruccionWrite(E0.a).

InstruccionNew → NEW E0;

InstruccionNew.a = instruccionNew(E0.a).

InstruccionDelete → DELETE E0;

InstruccionDelete.a = instruccionDelete(E0.a).

InstruccionCall → CALL IDENTIFICADOR PAR_ABIERTO ParametrosReales PAR_CERRADO;

InstruccionCall.a = instruccionCall(IDENTIFICADOR.lex, ParametrosReales.a)

ParametrosReales → ParametrosRealesAux;

ParametrosReales.a = ParametrosRealesAux.a.

ParametrosReales → ε;

ParametrosReales.a = paramReales_vacios().

ParametrosRealesAux → ParametrosRealesAux COMA E0;

ParametrosRealesAux.a = paramReales_muchos(ParametrosRealesAux.a, E0.a).

ParametrosRealesAux →E0;

ParametrosRealesAux.a = paramReales_uno(E0.a).

ExpresionBasica → NUM_ENT;

ExpresionBasica.a = num_ent(NUM_ENT.lex).

ExpresionBasica → NUM_REAL;

ExpresionBasica.a = num_real(NUM_REAL.lex).

ExpresionBasica → IDENTIFICADOR;

ExpresionBasica.a = id(IDENTIFICADOR.lex).

ExpresionBasica → BOOLEAN;

ExpresionBasica.a = boolean(BOOLEAN.lex).

ExpresionBasica → CADENA;

ExpresionBasica.a = cadena(CADENA.lex).

ExpresionBasica → NULL;

ExpresionBasica.a = null().

E0 → E1 MAS E0;

E0.a = mas(E1.a, E0.a).

E0 → E1 MENOS E1;

E0.a = Menos(E1.a, E1.a).

E0 → E1

E0.a = E1.a

E1 → E1 Op1 E2;

E1.a = exp(Op1.op, E1.a, E2.a).

E1 → E2;

E1.a = E2.a.

E2 →E2 Op2 E3;

E20.a = exp(Op2.op, E21.a, E3.a).

E2 →E3;

E2.a = E3.a.

E3 → E4 Op3 E4;

E3.a = exp(Op3.op, E40.a, E41.a).

E3 → E4;

E3.a = E4.a.

E4 → NOT E4 ;

E40.a = not(E41.a).

E4 → MENOS E5 ;

E4.a = menos(E5.a).

E4 → E5;

E4.a = E5.a.

E5 → E5 CORCHETE_ABIERTO E0 CORCHETE_CERRADO;

E50.a = indexacion(E51.a, E0.a).

E5 → E5 PUNTO IDENTIFICADOR;

E50.a = acceso_registro_punto(E51.a, IDENTIFICADOR.LEX).

E5 → E5 FLECHA IDENTIFICADOR;

E50.a = acceso_registro_flecha(E51.a, IDENTIFICADOR.LEX).

E5 → E6;

E5.a = E6.a.

E6 → MULT E6;

E60.a = asterisco(E61.a).

E6 → E7;

E6.a = E7.a.

E7 → ExpresionBasica;

E7.a = EpresionBasica.a.

E7 → PAR_ABIERTO E0 PAR_CERRADO;

E7.a = E0.a.

Op1 → AND;

Op1.op = ‘and’.

Op1 → OR;

Op1.op =’or’.

Op2 → MENOR;

Op2.op =’menor’.

Op2 → MAYOR;
Op2.op =’mayor’.
Op2 → MENOR_IGUAL;
Op2.op =’menor_igual’.
Op2 → MAYOR_IGUAL;
Op2.op =’mayor_igual’.
Op2 → DIFERENTE;
Op2.op =’diferente’.
Op2 → IGUAL;
Op2.op =’igual’.
Op3 → MULT;
Op3.op = ‘mult’.
Op3 → DIV;
Op3.op = ‘div’.
Op3 → PORCENTAJE;
Op3.op = ‘porcentaje’.
}

FUNCIONES SEMÁNTICAS:

```
fun exp(Op, Arg0, Arg1){
        switch Op
        case ‘and’:
                return and(Arg0,Arg1)
        case ‘or’:
                return or(Arg0,Arg1)
        case ‘menor’:
                return menor(Arg0,Arg1)
        case ‘mayor’:
                return mayor(Arg0,Arg1)
        case ’menor_igual’:
                return menor_igual(Arg0,Arg1)
        case ’mayor_igual’:
                return mayor_igual(Arg0,Arg1)
        case ’diferente’:
                return diferente(Arg0,Arg1)
        case ’igual’:
                return igual(Arg0,Arg1)
        case ‘mult’:
                return mult(Arg0,Arg1)
        case ‘div’:
                return div(Arg0,Arg1)
        case ‘porcentaje’
                return porcentaje(Arg0,Arg1)
}
```

3. Acondicionamiento de la gramática.

G = (Vn , Vt, P, S)

VN = {S, SecDec, Else, Declaraciones, DeclaracionesAux, Declaracion, DeclaracionVar, DeclaracionTipo, DeclaracionProc, ParametrosFormales, ParametrosFormalesAux, Parametro, ParametroAux, Bloque, Tipo, Array, Registro, Campos, CamposAux, Campo, Puntero, SecInstr, SecInstrAux, Instruccion, InstruccionAsignacion, InstruccionIf, SecInstr2, InstruccionWhile, InstruccionRead, InstruccionWrite, InstruccionNew, InstruccionDelete, InstruccionCall, ParametrosReales, ParametrosRealesAux, ExpresionBasica, E0, E1, E2, E3, E4, E5, E6, E7, Op1, Op2, Op3, Op4, E0_AUX,E1_AUX, E2_AUX, E3_AUX, E5_AUX}

VT = {NL, OF, TYPE, PROC, SEPARADOR, PUNTOCOMA, VAR, IDENTIFICADOR, PAR_ABIERTO, PAR_CERRADO, COMA, AMPERSAND, LLAVE_ABIERTA, LLAVE_CERRADA, INT, REAL, BOOL, STRING, ARRAY, RECORD, POINTER, CORCHETE_ABIERTO, CORCHETE_CERRADO, ASIGNACION, MAS, MENOS, NOT, NUM_ENT, NUM_REAL, BOOLEAN, AND, OR, MENOR, MAYOR, MENOR_IGUAL, MAYOR_IGUAL, DIFERENTE, IGUAL, MULT, DIV, NULL, IF, ELSE, WHILE, ENDIF, ENDWHILE, DO, THEN, READ, WRITE, NEW, DELETE, CALL, PORCENTAJE, PUNTO, FLECHA , CADENA}

P = {

SAux → S |-;

SAux.a = S.a.

S → SecDec SecInstr;

S.a = prog (secDec.a, SecInstr.a).

SecDec → Declaraciones SEPARADOR;

SecDec.a = Declaraciones.a.

SecDec → ε;

SecDes.a = decs_vacia().

Declaraciones → Declaracion DeclaracionesAux;

Declaraciones.a = DeclaracionesAux.a.

DeclaracionesAux.ah = decs_una(Declaracion.a).

DeclaracionesAux → PUNTOCOMA Declaracion DeclaracionesAux;

DeclaracionesAux0.a = DeclaracionesAux1.a.

DeclaracionesAux1.ah = decs_muchas(DeclaracionesAux0.ah, Declaracion.a).

DeclaracionesAux → ε;

DeclaracionesAux.a = DeclaracionesAux.ah.

Declaracion → DeclaracionVar;

Declaracion.a = DeclaracionVar.a.

Declaracion → DeclaracionTipo ;

Declaracion.a = DeclaracionTipo.a.

Declaracion → DeclaracionProc;

Declaracion.a = DeclaracionProc.a.

DeclaracionVar → VAR Tipo IDENTIFICADOR;

DeclaracionVar.a = dec_var(Tipo.a, IDENTIFICADOR.lex).

DeclaracionTipo → TYPE Tipo IDENTIFICADOR;

DeclaracionTipo.a = dec_tipo(Tipo.a, IDENTIFICADOR.lex).

DeclaracionProc → PROC IDENTIFICADOR PAR_ABIERTO ParametrosFormales PAR_CERRADO Bloque;

DeclaracionProc.a = dec_proc(IDENTIFICADOR.lex, ParametrosFormales.a, Bloque.a).

ParametrosFormales → Parametro ParametrosFormalesAux;

ParametrosFormales.a = ParametrosFormalesAux.a.

ParametrosFormalesAux.ah = paramForm_uno(Parametro.a).

ParametrosFormales → ε;

ParametrosFormales.a = paramForm_vacio().

ParametrosFormalesAux → COMA Parametro ParametrosFormalesAux;

ParametrosFormalesAux0.a = ParametrosFormalesAux1.a.

ParametrosFormalesAux1.ah = paramForm_muchos(ParametrosFormalesAux0.ah, Parametro.a).

ParametrosFormalesAux → ε;

ParametrosFormalesAux.a = ParametrosFormalesAux.ah.

Parametro → Tipo ParametroAux;

Parametro.a = ParametroAux.a.

ParametroAux.ah = Tipo.a.

ParametroAux → IDENTIFICADOR;

ParametroAux.a = param_sin_amp(ParametroAux.ah, IDENTIFICADOR.lex).

ParametroAux → AMPERSAND IDENTIFICADOR;

ParametroAux.a = param_con_amp(ParametroAux.ah, IDENTIFICADOR.lex).

Bloque → LLAVE_ABIERTA SecDec SecInstr LLAVE_CERRADA;

Bloque.a = bloque(SecDec.a, SecInstr.a).

Tipo → INT;

Tipo.a = tipoInt().

Tipo → REAL;

Tipo.a = tipoRealt().

Tipo → BOOL;

Tipo.a = tipoBool().

Tipo → STRING;

Tipo.a = tipoString().

Tipo → IDENTIFICADOR;

Tipo.a = tipoId(IDENTIFICADOR.lex).

Tipo → Array;

Tipo.a = array.a.

Tipo → Registro;

Tipo.a = resgistro.a.

Tipo → Puntero;

Tipo.a = puntero.a.

Array → ARRAY CORCHETE_ABIERTO NUM_ENT CORCHETE_CERRADO OF Tipo;

Array.a = tipoArray(NUN_ENT.lex, Tipo.a).

Registro → RECORD LLAVE_ABIERTA Campos LLAVE_CERRADA;

Registro.a = tipoRegistro(Campos.a).

Campos → Campo CamposAux;

Campos.a = CamposAux.a.

CamposAux.ah = campos_uno(Campo.a).

CamposAux → PUNTOCOMA Campo CamposAux;

CamposAux0.a = CamposAux1.a.

CamposAux1.ah = campos_muchos(CamposAux0.ah, Campo.a).

CamposAux → ε;

CamposAux.a = CamposAux.ah.

Campo → Tipo IDENTIFICADOR;

Campo.a = campo(Tipo.a, IDENTIFICADOR.lex).

Puntero → POINTER Tipo;

Puntero.a = tipoPuntero(Tipo.a).

SecInstr → Instruccion SecInstrAux;

SecInstr.a = SecInstrAux.a

SecInstrAux.ah = instr_una(Instruccion.a)

SecInstrAux → PUNTOCOMA Instruccion SecInstrAux;

SecInstrAux0.a = SecInstrAux1.a

SecInstrAux1.ah = instr_muchas(SecInstrAux0.ah, Instruccion.a)

SecInstrAux → ε;

SecInstrAux.a = SecInstrAux.ah

Instruccion → InstruccionAsignacion;

Instruccion.a = InstruccionAsig.a.

Instruccion → InstruccionIf ;

Instruccion.a = InstruccionIF.a.

Instruccion → InstruccionWhile ;

Instruccion.a = InstruccionWhile.a.

Instruccion → InstruccionRead ;

Instruccion.a = InstruccionRead.a.

Instruccion → InstruccionWrite ;

Instruccion.a = InstruccionWrite.a.

Instruccion → NL ;

Instruccion.a = InstruccionNL().

Instruccion → InstruccionNew ;

Instruccion.a = InstruccionNew.a.

Instruccion → InstruccionDelete ;

Instruccion.a = InstruccionDelete.a.

Instruccion → InstruccionCall ;

Instruccion.a = InstruccionCall.a.

Instruccion → Bloque;

Instruccion.a = instruccionBloque(Bloque.a).

InstruccionAsignacion → E0 ASIGNACION E0;

InstruccionAsignacion.a = instruccionAsig(E0.a, E0.a).

InstruccionIf → IF E0 THEN SecInstr2 InstruccionIf_aux;

InstruccionIf.a = InstruccionIf_aux.a.

InstruccionIf_aux.ah1 = E0.a.

InstruccionIf_aux.ah2 = SecInstr2.a.

InstruccionIf_aux → ENDIF;

InstruccionIf_aux.a = instruccionIf(InstruccionIf_aux.ah1, InstruccionIf_aux.ah2).

InstruccionIf_aux → ELSE SecInstr2 ENDIF;

InstruccionIf_aux.a = instruccionIfElse(InstruccionIf_aux.ah1, InstruccionIf_aux.ah2, SecInstr2.a).

SecInstr2 → SecInstr;

SecInstr2.a = SecInstr.a.

SecInstr2 → ε;

SecInstr2.a = instr_vacia().

InstruccionWhile → WHILE E0 DO SecInstr2 ENDWHILE;

InstruccionWhile.a = instruccionWhile(E0.a, SecInstr2.a).

InstruccionRead → READ E0;

InstruccionRead.a = instruccionRead(E0.a).

InstruccionWrite → WRITE E0;

InstruccionWrite.a = instruccionWrite(E0.a).

InstruccionNew → NEW E0;

InstruccionNew.a = instruccionNew(E0.a).

InstruccionDelete → DELETE E0;

InstruccionDelete.a = instruccionDelete(E0.a).

InstruccionCall → CALL IDENTIFICADOR PAR_ABIERTO ParametrosReales PAR_CERRADO;

InstruccionCall.a = instruccionCall(IDENTIFICADOR.lex, ParametrosReales.a).

ParametrosReales → E0 ParametrosRealesAux;

ParametrosReales.a = ParametrosRealesAux.a

ParametrosReales.ah = paramReales_uno(E0.a)

ParametrosReales → ε;

ParametrosReales.a = paramReales_vacios().

ParametrosRealesAux → COMA E0 ParametrosRealesAux;

ParametrosRealesAux0.a = ParametrosRealesAux1.a

ParametrosRealesAux1.ah = paramReales_muchos(ParametrosRealesAux0.ah, E0.a)

ParametrosRealesAux → ε;

ParametrosRealesAux.a = ParametrosRealesAux.ah

ExpresionBasica → NUM_ENT;

ExpresionBasica.a = num_ent(NUM_ENT.lex).

ExpresionBasica → NUM_REAL;

ExpresionBasica.a = num_real(NUM_REAL.lex).

ExpresionBasica → IDENTIFICADOR;

ExpresionBasica.a = id(IDENTIFICADOR.lex).

ExpresionBasica → BOOLEAN;

ExpresionBasica.a = boolean(BOOLEAN.lex).

ExpresionBasica → CADENA;

ExpresionBasica.a = cadena(CADENA.lex).

ExpresionBasica → NULL;

ExpresionBasica.a = null().

E0 → E1 E0_AUX ;

E0_AUX.ah = E1.a

E0.a = E0_AUX.a

E0_AUX → MAS E0;

E0_AUX.a = mas(E0_aux.ah, E0.a)

E0_AUX → MENOS E1;

E0_AUX.a = menos(E0_aux.ah, E1.a)

E0_AUX → ε;

E0_AUX.a = E0_AUX.ah.

E1 → E2 E1_AUX;

E1_AUX.ah = E2.a

E1.a = E1_AUX.a

E1_AUX → Op1 E2 E1_AUX;

E1_AUX0.a = E1_AUX1.a

E1_AUX1.ah = exp(Op1.op, E1_AUX0.ah, E2.a)

E1_AUX → ε;

E1_AUX.a = E1_AUX.ah.

E2 → E3 E2_AUX;

E2_AUX.ah = E3.a

E2.a = E2_AUX.a

E2_AUX → Op2 E3 E2_AUX ;

E2_AUX0.a = E2_AUX1.a

E2_AUX1.ah = exp (Op2.op, E2_AUX0.ah, E3.a)

E2_AUX →ε ;

E2_AUX.a = E2_AUX.ah.

E3 → E4 E3_AUX;

E3.a = E3_AUX.a

E3_AUX.ah = E4.a

E3_AUX → Op3 E4;

E3_AUX.a = exp(Op3.op,E3_AUX.ah, E4.a)

E3_AUX → ε;

E3_AUX.a = E3_AUX.ah

E4 → MENOS E5;

E4.a = negativo(E5.a)

E4 → NOT E4;

$E4_1.a = not(E4_2.a)$

E4 → E5;

E4.a = E5.a

E5 → E6 E5_AUX ;

E5.a = E5_AUX.a

E5_AUX.ah = E6.a

E5_AUX → CORCHETE_ABIERTO E0 CORCHETE_CERRADO E5_AUX;

E5_AUX0.a = E5_AUX1.a

E5_AUX1.ah = indexacion(E5_AUX0.ah, E0.a)

E5_AUX → PUNTO IDENTIFICADOR E5_AUX;

E5_AUX0.a = E5_AUX1.a

E5_AUX1.ah = acceso_registro_punto(E5_AUX0.ah, IDENTIFICADOR.lex)

E5_AUX → FLECHA IDENTIFICADOR E5_AUX;

E5_AUX0.a = E5_AUX1.a

E5_AUX1.ah = acceso_registro_flecha(E5_AUX0.ah, IDENTIFICADOR.lex)

E5_AUX → ε

E5_AUX.a = E5_AUX.ah

E6 → MULT E6;

E6.a = asterisco(E6.a)

E6 → E7

E6.a = E7.a

E7 → ExpresionBasica;

E7.a = Expresionbasica.a

E7 → PAR_ABIERTO E0 PAR_CERRADO;

E7.a = E0.a

Op1 → AND;

Op1.op = 'and'.

Op1 → OR;

Op1.op ='or'.

Op2 → MENOR;

Op2.op ='menor'.

Op2 → MAYOR;

Op2.op ='mayor'.

Op2 → MENOR_IGUAL;

Op2.op ='menor_igual'.

Op2 → MAYOR_IGUAL;

Op2.op ='mayor_igual'.

Op2 → DIFERENTE;

Op2.op ='diferente'.

Op2 → IGUAL;

Op2.op ='igual'.

Op3 → MULT;

Op3.op = 'mult'.

Op3 → DIV;

Op3.op = 'div'.

Op3 → PORCENTAJE;

Op3.op = 'porcentaje'.

}