
Procesadores de Lenguajes

Memoria de proyecto — Hito 3:ASTs

GRUPO 14

RODRIGO SOUTO SANTOS
LEONARDO PRADO DE SOUZA
JUAN ANDRÉS HIBJAN CARDONA
IZAN RODRIGO SANZ

Grado en Ingeniería informática
Facultad de Informática
Universidad Complutense de Madrid



Índice general

1. Especificación de la Sintaxis Abstracta	2
1.1. Géneros de nodos	2
1.2. Funciones constructoras de nodos	2
1.2.1. Declaraciones	2
1.2.2. Tipos	2
1.2.3. Instrucciones	3
1.2.4. Expresiones	3
2. Especificación del constructor de ASTs	4
2.1. Gramática s-atribuida	4
2.1.1. Declaraciones	4
2.1.2. Tipos	4
2.1.3. Instrucciones	5
2.1.4. Expresiones	6
2.1.5. Operadores	7
2.2. Funciones semánticas auxiliares	7
2.2.1. Expresiones binarias	7
3. Acondicionamiento del constructor de ASTs	8
3.1. Gramática s-atribuida	8
3.1.1. Declaraciones	8
3.1.2. Tipos	9
3.1.3. Instrucciones	9
3.1.4. Expresiones	10
3.1.5. Operadores	12
3.2. Funciones semánticas auxiliares	12
3.2.1. Expresiones binarias	12
4. Especificación del procesamiento ‘impresión bonita’	13
4.1. Funciones de procesamiento	13
4.1.1. Declaraciones	13
4.1.2. Tipos	14
4.1.3. Instrucciones	15
4.1.4. Expresiones	16
4.1.5. Funciones auxiliares	18

1 | Especificación de la Sintaxis Abstracta

1.1. Géneros de nodos

Bloq
SecDecs, LDecs, Dec
SecIs, LIs, I
ParamFs, LParamFs, ParamF
ParamRs, LParamRs
LCampos
TipoNom, Tipo
Exp

1.2. Funciones constructoras de nodos

bloque : *SecDecs* \times *SecIs* \longrightarrow *Bloq*

1.2.1. Declaraciones

si_decs : *LDecs* \longrightarrow *SecDecs*
no_decs : \longrightarrow *SecDecs*
muchas_decs : *LDecs* \times *Dec* \longrightarrow *LDecs*
una_dec : *Dec* \longrightarrow *LDecs*
dec_base : *TipoNom* \longrightarrow *Dec*
dec_type : *TipoNom* \longrightarrow *Dec*
dec_proc : **string** \times *ParamFs* \times *Bloq* \longrightarrow *Dec*
si_params_f : *LParamFs* \longrightarrow *ParamFs*
no_params_f : \longrightarrow *ParamFs*
muchos_params_f : *LParamFs* \times *ParamF* \longrightarrow *LParamFs*
un_param_f : *ParamF* \longrightarrow *LParamFs*
si_refparam_f : *Tipo* \times **string** \longrightarrow *ParamF*
no_refparam_f : *Tipo* \times **string** \longrightarrow *ParamF*

1.2.2. Tipos

tipo_nombre : *Tipo* \times **string** \longrightarrow *TipoNom*
tipo_array : *Tipo* \times **string** \longrightarrow *Tipo*
tipo_indir : *Tipo* \longrightarrow *Tipo*
tipo_struct : *LCampos* \longrightarrow *Tipo*
tipo_int : \longrightarrow *Tipo*
tipo_real : \longrightarrow *Tipo*
tipo_bool : \longrightarrow *Tipo*
tipo_string : \longrightarrow *Tipo*
tipo_type : **string** \longrightarrow *Tipo*
muchos_campos : *LCampos* \times *TipoNom* \longrightarrow *LCampos*
un_campo : *TipoNom* \longrightarrow *LCampos*

1.2.3. Instrucciones

$si_ins : LIs \longrightarrow SecIs$
 $no_ins : \longrightarrow SecIs$
 $muchas_ins : LIs \times I \longrightarrow LIs$
 $una_ins : I \longrightarrow LIs$
 $ins_eval : Exp \longrightarrow I$
 $ins_if : Exp \times Bloq \longrightarrow I$
 $ins_if_else : I \times Bloq \longrightarrow I$
 $ins_while : Exp \times Bloq \longrightarrow I$
 $ins_read : Exp \longrightarrow I$
 $ins_write : Exp \longrightarrow I$
 $ins_nl : \longrightarrow I$
 $ins_new : Exp \longrightarrow I$
 $ins_delete : Exp \longrightarrow I$
 $ins_call : \mathbf{string} \times ParamRs \longrightarrow I$
 $ins_bloque : Bloq \longrightarrow I$
 $si_params_r : LParamRs \longrightarrow ParamRs$
 $no_params_r : \longrightarrow ParamRs$
 $muchos_params_r : LParamRs \times Exp \longrightarrow LParamRs$
 $un_param_r : Exp \longrightarrow LParamRs$

1.2.4. Expresiones

$exp_asig : Exp \times Exp \longrightarrow Exp$
 $exp_menor : Exp \times Exp \longrightarrow Exp$
 $exp_menor_ig : Exp \times Exp \longrightarrow Exp$
 $exp_mayor : Exp \times Exp \longrightarrow Exp$
 $exp_mayor_ig : Exp \times Exp \longrightarrow Exp$
 $exp_ig : Exp \times Exp \longrightarrow Exp$
 $exp_dist : Exp \times Exp \longrightarrow Exp$
 $exp_suma : Exp \times Exp \longrightarrow Exp$
 $exp_resta : Exp \times Exp \longrightarrow Exp$
 $exp_and : Exp \times Exp \longrightarrow Exp$
 $exp_or : Exp \times Exp \longrightarrow Exp$
 $exp_mul : Exp \times Exp \longrightarrow Exp$
 $exp_div : Exp \times Exp \longrightarrow Exp$
 $exp_mod : Exp \times Exp \longrightarrow Exp$
 $exp_menos : Exp \longrightarrow Exp$
 $exp_not : Exp \longrightarrow Exp$
 $exp_index : Exp \times Exp \longrightarrow Exp$
 $exp_reg : Exp \times \mathbf{string} \longrightarrow Exp$
 $exp_indir : Exp \longrightarrow Exp$
 $exp_entero : \mathbf{string} \longrightarrow Exp$
 $exp_real : \mathbf{string} \longrightarrow Exp$
 $exp_true : \longrightarrow Exp$
 $exp_false : \longrightarrow Exp$
 $exp_cadena : \mathbf{string} \longrightarrow Exp$
 $exp_iden : \mathbf{string} \longrightarrow Exp$
 $exp_null : \longrightarrow Exp$

2 | Especificación del constructor de ASTs

2.1. Gramática s-atribuida

$\text{programa} \rightarrow \text{bloque}$
 $\text{programa.a} = \text{bloque.a}$
 $\text{bloque} \rightarrow \{\text{seccion_declaraciones_opt} \text{ seccion_instrucciones_opt}\}$
 $\text{bloque.a} = \text{bloq}(\text{seccion_declaraciones_opt.a}, \text{seccion_instrucciones_opt.a})$

2.1.1. Declaraciones

$\text{seccion_declaraciones_opt} \rightarrow \text{seccion_declaraciones} \&\&$
 $\text{seccion_declaraciones_opt.a} = \text{si_decs}(\text{seccion_declaraciones.a})$
 $\text{seccion_declaraciones_opt} \rightarrow \epsilon$
 $\text{seccion_declaraciones_opt.a} = \text{no_decs}()$
 $\text{seccion_declaraciones} \rightarrow \text{seccion_declaraciones} ; \text{declaracion}$
 $\text{seccion_declaraciones}_0.\text{a} = \text{muchas_decs}(\text{seccion_declaraciones}_1.\text{a}, \text{declaracion.a})$
 $\text{seccion_declaraciones} \rightarrow \text{declaracion}$
 $\text{seccion_declaraciones.a} = \text{una_dec}(\text{declaracion.a})$
 $\text{declaracion} \rightarrow \text{tipo_nombre}$
 $\text{declaracion.a} = \text{dec_base}(\text{tipo_nombre.a})$
 $\text{declaracion} \rightarrow \text{type } \text{tipo_nombre}$
 $\text{declaracion.a} = \text{dec_type}(\text{tipo_nombre.a})$
 $\text{declaracion} \rightarrow \text{proc } \text{identificador } \text{parametros_formales } \text{bloque}$
 $\text{declaracion.a} = \text{dec_proc}(\text{identificador.lex}, \text{parametros_formales.a}, \text{bloque.a})$
 $\text{parametros_formales} \rightarrow (\text{lista_parametros_opt})$
 $\text{parametros_formales.a} = \text{lista_parametros_opt.a}$
 $\text{lista_parametros_opt} \rightarrow \text{lista_parametros}$
 $\text{lista_parametros_opt.a} = \text{si_params_f}(\text{lista_parametros.a})$
 $\text{lista_parametros_opt} \rightarrow \epsilon$
 $\text{lista_parametros_opt.a} = \text{no_params_f}()$
 $\text{lista_parametros} \rightarrow \text{lista_parametros} , \text{parametro}$
 $\text{lista_parametros}_0.\text{a} = \text{muchos_params_f}(\text{lista_parametros}_1.\text{a}, \text{parametro.a})$
 $\text{lista_parametros} \rightarrow \text{parametro}$
 $\text{lista_parametros.a} = \text{un_param_f}(\text{parametro.a})$
 $\text{parametro} \rightarrow \text{tipo} \& \text{identificador}$
 $\text{parametro.a} = \text{si_refparam_f}(\text{tipo.a}, \text{identificador.lex})$
 $\text{parametro} \rightarrow \text{tipo } \text{identificador}$
 $\text{parametro.a} = \text{no_refparam_f}(\text{tipo.a}, \text{identificador.lex})$

2.1.2. Tipos

$\text{tipo_nombre} \rightarrow \text{tipo } \text{identificador}$
 $\text{tipo_nombre.a} = \text{tipo_nombre}(\text{tipo.a}, \text{identificador.lex})$
 $\text{tipo} \rightarrow \text{tipo0}$
 $\text{tipo.a} = \text{tipo0.a}$
 $\text{tipo0} \rightarrow \text{tipo0} [\text{literalEntero}]$
 $\text{tipo0}_0.\text{a} = \text{tipo_array}(\text{tipo0}_1.\text{a}, \text{literalEntero.lex})$
 $\text{tipo0} \rightarrow \text{tipo1}$
 $\text{tipo0.a} = \text{tipo1.a}$
 $\text{tipo1} \rightarrow \wedge \text{tipo1}$
 $\text{tipo1}_0.\text{a} = \text{tipo_indir}(\text{tipo1}_1.\text{a})$
 $\text{tipo1} \rightarrow \text{tipo_base}$
 $\text{tipo1.a} = \text{tipo_base.a}$

$tipo_base \rightarrow \mathbf{struct} \{lista_campos\}$
 $tipo_base.a = tipo_struct(lista_campos.a)$
 $tipo_base \rightarrow \mathbf{int}$
 $tipo_base.a = tipo_int()$
 $tipo_base \rightarrow \mathbf{real}$
 $tipo_base.a = tipo_real()$
 $tipo_base \rightarrow \mathbf{bool}$
 $tipo_base.a = tipo_bool()$
 $tipo_base \rightarrow \mathbf{string}$
 $tipo_base.a = tipo_string()$
 $tipo_base \rightarrow \mathbf{identificador}$
 $tipo_base.a = tipo_type(identificador.lex)$
 $lista_campos \rightarrow lista_campos, tipo_nombre$
 $lista_campos_0.a = muchos_campos(lista_campos_1.a, tipo_nombre.a)$
 $lista_campos \rightarrow tipo_nombre$
 $lista_campos.a = un_campo(tipo_nombre.a)$

2.1.3. Instrucciones

$seccion_instrucciones_opt \rightarrow seccion_instrucciones$
 $seccion_instrucciones_opt.a = si_ins(seccion_instrucciones.a)$
 $seccion_instrucciones_opt \rightarrow \epsilon$
 $seccion_instrucciones_opt.a = no_ins()$
 $seccion_instrucciones \rightarrow lista_instrucciones$
 $seccion_instrucciones.a = lista_instrucciones.a$
 $lista_instrucciones \rightarrow lista_instrucciones; instruccion$
 $lista_instrucciones_0.a = muchas_ins(lista_instrucciones_1.a, instruccion.a)$
 $lista_instrucciones \rightarrow instruccion$
 $lista_instrucciones.a = una_ins(instruccion.a)$
 $instruccion \rightarrow @ expresion$
 $instruccion.a = ins_eval(expresion.a)$
 $instruccion \rightarrow \mathbf{if_ins}$
 $instruccion.a = if_ins.a$
 $instruccion \rightarrow \mathbf{if_ins} \mathbf{else} \mathbf{bloque}$
 $instruccion.a = ins_if_else(if_ins.a, bloque_1.a)$
 $instruccion \rightarrow \mathbf{while} \mathbf{expresion} \mathbf{bloque}$
 $instruccion.a = ins_while(expresion.a, bloque.a)$
 $instruccion \rightarrow \mathbf{read} \mathbf{expresion}$
 $instruccion.a = ins_read(expresion.a)$
 $instruccion \rightarrow \mathbf{write} \mathbf{expresion}$
 $instruccion.a = ins_write(expresion.a)$
 $instruccion \rightarrow \mathbf{nl}$
 $instruccion.a = ins_nl()$
 $instruccion \rightarrow \mathbf{new} \mathbf{expresion}$
 $instruccion.a = ins_new(expresion.a)$
 $instruccion \rightarrow \mathbf{delete} \mathbf{expresion}$
 $instruccion.a = ins_delete(expresion.a)$
 $instruccion \rightarrow \mathbf{call} \mathbf{identificador} \mathbf{parametros_reales}$
 $instruccion.a = ins_call(identificador.lex, parametros_reales.a)$
 $instruccion \rightarrow \mathbf{bloque}$
 $instruccion.a = ins_bloque(bloque.a)$
 $\mathbf{if_ins} \rightarrow \mathbf{if} \mathbf{expresion} \mathbf{bloque}$
 $\mathbf{if_ins}.a = ins_if(expresion.a, bloque.a)$
 $\mathbf{parametros_reales} \rightarrow (lista_expresiones_opt)$
 $\mathbf{parametros_reales}.a = lista_expresiones_opt.a$
 $lista_expresiones_opt \rightarrow lista_expresiones$
 $lista_expresiones_opt.a = si_params_r(lista_expresiones.a)$
 $lista_expresiones_opt \rightarrow \epsilon$
 $lista_expresiones_opt.a = no_params_r()$
 $lista_expresiones \rightarrow lista_expresiones, expresion$

$lista_expresiones_0.a = muchos_params_r(lista_expresiones_1.a, expresion.a)$
 $lista_expresiones \rightarrow expresion$
 $lista_expresiones.a = un_param_r(expresion.a)$

2.1.4. Expresiones

$expresion \rightarrow E0$
 $expresion.a = E0.a$
 $E0 \rightarrow E1 = E0$
 $E0_0.a = mkopbin("=", E1.a, E0_1.a)$
 $E0 \rightarrow E1$
 $E0.a = E1.a$
 $E1 \rightarrow E1 \text{ op_relacional } E2$
 $E1_0.a = mkopbin(op_relacional.op, E1_1.a, E2.a)$
 $E1 \rightarrow E2$
 $E1.a = E2.a$
 $E2 \rightarrow E2 + E3$
 $E2_0.a = mkopbin("+", E2_1.a, E3.a)$
 $E2 \rightarrow E3 - E3$
 $E2.a = mkopbin("-", E3_0.a, E3_1.a)$
 $E2 \rightarrow E3$
 $E2.a = E3.a$
 $E3 \rightarrow E4 \text{ and } E3$
 $E3_0.a = mkopbin("and", E4.a, E3_1.a)$
 $E3 \rightarrow E4 \text{ or } E4$
 $E3.a = mkopbin("or", E4_0.a, E4_1.a)$
 $E3 \rightarrow E4$
 $E3.a = E4.a$
 $E4 \rightarrow E4 \text{ op_mult } E5$
 $E4_0.a = mkopbin(op_mult.op, E4_1.a, E5.a)$
 $E4 \rightarrow E5$
 $E4.a = E5.a$
 $E5 \rightarrow - E5$
 $E5_0.a = exp_menos(E5.a)$
 $E5 \rightarrow \text{not } E5$
 $E5_0.a = exp_not(E5.a)$
 $E5 \rightarrow E6$
 $E5.a = E6.a$
 $E6 \rightarrow E6 [expresion]$
 $E6_0.a = exp_index(E6_1.a, expresion.a)$
 $E6 \rightarrow E6 . \text{identificador}$
 $E6_0.a = exp_ref(E6_1.a, identificador.lex)$
 $E6 \rightarrow E6^{\wedge}$
 $E6_0.a = exp_indir(E6_1.a)$
 $E6 \rightarrow E7$
 $E6.a = E7.a$
 $E7 \rightarrow expresion_basica$
 $E7.a = expresion_basica.a$
 $E7 \rightarrow (E0)$
 $E7.a = E0.a$
 $expresion_basica \rightarrow \text{literalEntero}$
 $expresion_basica.a = exp_entero(literalEntero.lex)$
 $expresion_basica \rightarrow \text{literalReal}$
 $expresion_basica.a = exp_real(literalReal.lex)$
 $expresion_basica \rightarrow \text{true}$
 $expresion_basica.a = exp_true()$
 $expresion_basica \rightarrow \text{false}$
 $expresion_basica.a = exp_false()$
 $expresion_basica \rightarrow \text{literalCadena}$
 $expresion_basica.a = exp_cadena(literalCadena.lex)$

```

expresion_basica → identificador
    expresion_basica.a = exp_iden(identificador.lex)
expresion_basica → null
    expresion_basica.a = exp_null()

```

2.1.5. Operadores

```

op_relacional → <
    op_relacional.op = "<"
op_relacional → <=
    op_relacional.op = "<="
op_relacional → >
    op_relacional.op = ">"
op_relacional → >=
    op_relacional.op = ">="
op_relacional → ==
    op_relacional.op = "=="
op_relacional → !=
    op_relacional.op = "!="
op_mult → *
    op_mult.op = "*"
op_mult → /
    op_mult.op = "/"
op_mult → %
    op_mult.op = "%"

```

2.2. Funciones semánticas auxiliares

2.2.1. Expresiones binarias

```

fun mkopbin(op, opnd1, opnd2) :
    op = "=" → return exp_asig(opnd1, opnd2)
    op = "<" → return exp_menor(opnd1, opnd2)
    op = "<=" → return exp_menor_ig(opnd1, opnd2)
    op = ">" → return exp_mayor(opnd1, opnd2)
    op = ">=" → return exp_mayor_ig(opnd1, opnd2)
    op = "==" → return exp_ig(opnd1, opnd2)
    op = "!=" → return exp_dist(opnd1, opnd2)
    op = "+" → return exp_suma(opnd1, opnd2)
    op = "-" → return exp_resta(opnd1, opnd2)
    op = "and" → return exp_and(opnd1, opnd2)
    op = "or" → return exp_or(opnd1, opnd2)
    op = "*" → return exp_mul(opnd1, opnd2)
    op = "/" → return exp_div(opnd1, opnd2)
    op = "%" → return exp_mod(opnd1, opnd2)

```


3 | Acondicionamiento del constructor de ASTs

3.1. Gramática s-atribuida

$programa \rightarrow bloque$
 $programa.a = bloque.a$
 $bloque \rightarrow \{seccion_declaraciones_opt\ seccion_instrucciones_opt\}$
 $bloque.a = bloq(seccion_declaraciones_opt.a, seccion_instrucciones_opt.a)$

3.1.1. Declaraciones

$seccion_declaraciones_opt \rightarrow seccion_declaraciones \&\&$
 $seccion_declaraciones_opt.a = si_decs(seccion_declaraciones.a)$
 $seccion_declaraciones_opt \rightarrow \epsilon$
 $seccion_declaraciones_opt.a = no_decs()$
 $seccion_declaraciones \rightarrow declaracion\ resto_sd$
 $resto_sd.ah = una_dec(declaracion.a)$
 $seccion_declaraciones_opt.a = resto_sd.a$
 $resto_sd \rightarrow ;\ declaracion\ resto_sd$
 $resto_sd_1.ah = muchas_decs(resto_sd_0.ah, declaracion.a)$
 $resto_sd_0.a = resto_sd_1.a$
 $resto_sd \rightarrow \epsilon$
 $resto_sd.a = resto_sd.ah$
 $declaracion \rightarrow tipo_nombre$
 $declaracion.a = dec_base(tipo_nombre.a)$
 $declaracion \rightarrow \mathbf{type}\ tipo_nombre$
 $declaracion.a = dec_type(tipo_nombre.a)$
 $declaracion \rightarrow \mathbf{proc}\ \mathbf{identificador}\ parametros_formales\ bloque$
 $declaracion.a = dec_proc(\mathbf{identificador.lex}, parametros_formales.a, bloque.a)$
 $parametros_formales \rightarrow (lista_parametros_opt)$
 $parametros_formales.a = lista_parametros_opt.a$
 $lista_parametros_opt \rightarrow lista_parametros$
 $lista_parametros_opt.a = si_params_f(lista_parametros.a)$
 $lista_parametros_opt \rightarrow \epsilon$
 $lista_parametros_opt.a = no_params_f()$
 $lista_parametros \rightarrow parametro\ resto_lp$
 $resto_lp.ah = un_param_f(parametro.a)$
 $lista_parametros.a = resto_lp.a$
 $resto_lp \rightarrow ,\ parametro\ resto_lp$
 $resto_lp_1.ah = muchos_params_f(resto_lp_0.ah, parametro.a)$
 $resto_lp_0.a = resto_lp_1.a$
 $resto_lp \rightarrow \epsilon$
 $resto_lp.a = resto_lp_1.ah$
 $parametro \rightarrow tipo\ resto_parametro$
 $resto_parametro.ah = tipo.a$
 $parametro.a = resto_parametro.a$
 $resto_parametro \rightarrow \&\ \mathbf{identificador}$
 $resto_parametro.a = si_refparam_f(resto_parametro.ah, \mathbf{identificador.lex})$
 $resto_parametro \rightarrow \mathbf{identificador}$
 $resto_parametro.a = no_refparam_f(resto_parametro.ah, \mathbf{identificador.lex})$

3.1.2. Tipos

$tipo_nombre \rightarrow tipo_identificador$
 $tipo_nombre.a = tipo_nombre(tipo.a, identificador.lex)$
 $tipo \rightarrow tipo0$
 $tipo.a = tipo0.a$
 $tipo0 \rightarrow tipo1\ resto_tipo0$
 $resto_tipo0.ah = tipo1.a$
 $tipo0.a = resto_tipo0.a$
 $resto_tipo0 \rightarrow [literalEntero] resto_tipo0$
 $resto_tipo0_1.ah = tipo_array(tipo0_0.ah, literalEntero.lex)$
 $tipo0_0.a = tipo0_1.a$
 $resto_tipo0 \rightarrow \epsilon$
 $tipo0.a = tipo0.ah$
 $tipo1 \rightarrow \wedge tipo1$
 $tipo1_0.a = tipo_indir(tipo1_1.a)$
 $tipo1 \rightarrow tipo_base$
 $tipo1.a = tipo_base.a$
 $tipo_base \rightarrow \mathbf{struct} \{lista_campos\}$
 $tipo_base.a = tipo_struct(lista_campos.a)$
 $tipo_base \rightarrow \mathbf{int}$
 $tipo_base.a = tipo_int()$
 $tipo_base \rightarrow \mathbf{real}$
 $tipo_base.a = tipo_real()$
 $tipo_base \rightarrow \mathbf{bool}$
 $tipo_base.a = tipo_bool()$
 $tipo_base \rightarrow \mathbf{string}$
 $tipo_base.a = tipo_string()$
 $tipo_base \rightarrow \mathbf{identificador}$
 $tipo_base.a = tipo_type(identificador.lex)$
 $lista_campos \rightarrow tipo_nombre resto_lc$
 $resto_lc.ah = un_campo(tipo_nombre.a)$
 $lista_campos.a = resto_lc.a$
 $resto_lc \rightarrow , tipo_nombre resto_lc$
 $resto_lc_1.ah = muchos_campos(resto_lc_0.ah, tipo_nombre.a)$
 $resto_lc_0.a = resto_lc_0.a$
 $resto_lc \rightarrow \epsilon$
 $resto_lc.a = resto_lc.ah$

3.1.3. Instrucciones

$seccion_instrucciones_opt \rightarrow seccion_instrucciones$
 $seccion_instrucciones_opt.a = si_ins(seccion_instrucciones.a)$
 $seccion_instrucciones_opt \rightarrow \epsilon$
 $sseccion_instrucciones_opt.a = no_ins()$
 $seccion_instrucciones \rightarrow lista_instrucciones$
 $seccion_instrucciones.a = lista_instrucciones.a$
 $lista_instrucciones \rightarrow instruccion resto_li$
 $resto_li.ah = una_ins(instruccion.a)$
 $lista_instrucciones.a = resto_li.a$
 $resto_li \rightarrow ; instruccion resto_li$
 $resto_li_1.ah = muchas_ins(resto_li_0.ah, instruccion.a)$
 $resto_li_0.a = resto_li_1.a$
 $resto_li \rightarrow \epsilon$
 $resto_li.a = resto_li.ah$
 $instruccion \rightarrow @ expresion$
 $instruccion.a = ins_eval(expresion.a)$
 $instruccion \rightarrow if_ins resto_ii$
 $resto_ii.ah = ins_if(expresion.a, bloque.a)$
 $instruccion.a = resto_ii.a$

```

resto_ii  $\rightarrow$  else bloque
    resto_ii.a = ins_if_else(resto_ii.ah, bloque1.a)
resto_ii  $\rightarrow$   $\epsilon$ 
    resto_ii.a = resto_ii.ah
instruccion  $\rightarrow$  while expresion bloque
    instruccion.a = ins_while(expresion.a, bloque.a)
instruccion  $\rightarrow$  read expresion
    instruccion.a = ins_read(expresion.a)
instruccion  $\rightarrow$  write expresion
    instruccion.a = ins_write(expresion.a)
instruccion  $\rightarrow$  nl
    instruccion.a = ins_nl()
instruccion  $\rightarrow$  new expresion
    instruccion.a = ins_new(expresion.a)
instruccion  $\rightarrow$  delete expresion
    instruccion.a = ins_delete(expresion.a)
instruccion  $\rightarrow$  call identificador parametros_reales
    instruccion.a = ins_call(identificador.lex, parametros_reales.a)
instruccion  $\rightarrow$  bloque
    instruccion.a = ins_bloque(bloque.a)
if_ins  $\rightarrow$  if expresion bloque
    if_ins.a = ins_if(expresion.a, bloque.a)
parametros_reales  $\rightarrow$  (lista_expresiones_opt)
    parametros_reales.a = lista_expresiones_opt.a
lista_expresiones_opt  $\rightarrow$  lista_expresiones
    lista_expresiones_opt.a = si_params_r(lista_expresiones.a)
lista_expresiones_opt  $\rightarrow$   $\epsilon$ 
    lista_expresiones_opt.a = no_params_r()
lista_expresiones  $\rightarrow$  expresion resto_le
    resto_le.ah = un_param_r(expresion.a)
    lista_expresiones.a = resto_le.a
resto_le  $\rightarrow$  , expresion resto_le
    resto_le1.ah = muchos_params_r(resto_le0.ah, expresion.a)
    resto_le0.a = resto_le1.a
resto_le  $\rightarrow$   $\epsilon$ 
    resto_le.a = resto_le.ah

```

3.1.4. Expresiones

```

expresion  $\rightarrow$  E0
    expresion.a = E0.a
E0  $\rightarrow$  E1 resto_E0
    resto_E0.ah = E1.a
    E0.a = resto_E0.a
resto_E0  $\rightarrow$  = E0
    resto_E0.a = mkopbin("=", resto_E0.ah, E0.a)
resto_E0  $\rightarrow$   $\epsilon$ 
    resto_E0.a = resto_E0.ah
E1  $\rightarrow$  E2 resto_E1
    resto_E1.ah = E2.a
    E1.a = resto_E1.a
resto_E1  $\rightarrow$  op_relacional E2 resto_E1
    resto_E11.ah = mkopbin(op_relacional.op, resto_E10.ah, E2.a)
    resto_E10.a = resto_E11.a
resto_E1  $\rightarrow$   $\epsilon$ 
    resto_E1.a = resto_E1.ah
E2  $\rightarrow$  E3 resto_E2_F resto_E2_R
    resto_E2_F.ah = E3.a
    resto_E2_R.ah = resto_E2_F.a
    E2.a = resto_E2_R.a

```

```

resto_E2_R  $\longrightarrow$  + E3 resto_E2_R
    resto_E2_R1.ah = mkopbin(" + ", resto_E2_R0.ah, E3.a)
    resto_E2_R0.a = resto_E2_R1.a
resto_E2_R  $\longrightarrow$   $\epsilon$ 
    resto_E2_R.a = resto_E2_R.ah
resto_E2_F  $\longrightarrow$  - E3
    resto_E2_F.a = mkopbin(" - ", resto_E2_F.ah, E3.a)
resto_E2_F  $\longrightarrow$   $\epsilon$ 
    resto_E2_F.a = resto_E2_F.ah
E3  $\longrightarrow$  E4 resto_E3
    resto_E3.ah = E4.a
    E3.a = resto_E3.a
resto_E3  $\longrightarrow$  and E3
    resto_E3.a = mkopbin("and", resto_E3.ah, E3.a)
resto_E3  $\longrightarrow$  or E4
    resto_E3.a = mkopbin("or", resto_E3.ah, E4.a)
resto_E3  $\longrightarrow$   $\epsilon$ 
    resto_E3.a = resto_E3.ah
E4  $\longrightarrow$  E5 resto_E4
    resto_E4.ah = E5.a
    E4.a = resto_E4.a
resto_E4  $\longrightarrow$  op_mult E5 resto_E4
    resto_E41.ah = mkopbin(op_mult.op, resto_E40.ah, E5.a)
    resto_E40.a = resto_E41.a
resto_E4  $\longrightarrow$   $\epsilon$ 
    resto_E4.a = resto_E4.ah
E5  $\longrightarrow$  - E5
    E50.a = mkopun(" - ", E51.a)
E5  $\longrightarrow$  not E5
    E50.a = mkopun("not", E51.a)
E5  $\longrightarrow$  E6
    E5.a = E6.a
E6  $\longrightarrow$  E7 resto_E6
    resto_E6.ah = E7.a
    E6.a = resto_E6.a
resto_E6  $\longrightarrow$  [expresion] resto_E6
    resto_E61.ah = exp_index(resto_E60.ah, expresion.a)
    resto_E60.a = resto_E61.a
resto_E6  $\longrightarrow$  . identificador resto_E6
    resto_E61.ah = exp_ref(resto_E60.ah, identificador.lex)
    resto_E60.a = resto_E61.a
resto_E6  $\longrightarrow$  ^ resto_E6
    resto_E61.ah = exp_indir(resto_E60.ah)
    resto_E60.a = resto_E61.a
resto_E6  $\longrightarrow$   $\epsilon$ 
    resto_E6.a = resto_E6.ah
E7  $\longrightarrow$  expresion_basica
    E7.a = expresion_basica.a
E7  $\longrightarrow$  (E0)
    E7.a = E0.a
expresion_basica  $\longrightarrow$  literalEntero
    expresion_basica.a = exp_entero(literalEntero.lex)
expresion_basica  $\longrightarrow$  literalReal
    expresion_basica.a = exp_real(literalReal.lex)
expresion_basica  $\longrightarrow$  true
    expresion_basica.a = exp_true()
expresion_basica  $\longrightarrow$  false
    expresion_basica.a = exp_false()
expresion_basica  $\longrightarrow$  literalCadena
    expresion_basica.a = exp_cadena(literalCadena.lex)

```

```

expresion_basica → identificador
    expresion_basica.a = exp_iden(identificador.lex)
expresion_basica → null
    expresion_basica.a = exp_null()

```

3.1.5. Operadores

```

op_relacional → <
    op_relacional.op = "<"
op_relacional → <=
    op_relacional.op = "<="
op_relacional → >
    op_relacional.op = ">"
op_relacional → >=
    op_relacional.op = ">="
op_relacional → ==
    op_relacional.op = "=="
op_relacional → !=
    op_relacional.op = "!="
op_mult → *
    op_mult.op = "*"
op_mult → /
    op_mult.op = "/"
op_mult → %
    op_mult.op = "%"

```

3.2. Funciones semánticas auxiliares

3.2.1. Expresiones binarias

```

fun mkopbin(op, opnd1, opnd2) :
    op = "=" → return exp_asig(opnd1, opnd2)
    op = "<" → return exp_menor(opnd1, opnd2)
    op = "<=" → return exp_menor_ig(opnd1, opnd2)
    op = ">" → return exp_mayor(opnd1, opnd2)
    op = ">=" → return exp_mayor_ig(opnd1, opnd2)
    op = "==" → return exp_ig(opnd1, opnd2)
    op = "!=" → return exp_dist(opnd1, opnd2)
    op = "+" → return exp_suma(opnd1, opnd2)
    op = "-" → return exp_resta(opnd1, opnd2)
    op = "and" → return exp_and(opnd1, opnd2)
    op = "or" → return exp_or(opnd1, opnd2)
    op = "*" → return exp_mul(opnd1, opnd2)
    op = "/" → return exp_div(opnd1, opnd2)
    op = "%" → return exp_mod(opnd1, opnd2)

```

4 | Especificación del procesamiento ‘impresión bonita’

Imprime los tokens del programa leído, uno en cada línea, pero omitiendo los paréntesis redundantes en las expresiones (es decir, aquellos paréntesis que, eliminados, no cambian el significado de dichas expresiones). Las palabras reservadas se escribirán con todas las letras en minúscula, y entre ángulos (<...>). El fin de fichero se escribirá como <EOF>.

4.1. Funciones de procesamiento

```
imprime(bloque(SecDecs, SecIs)) :
    println("{")
    imprime(SecDecs)
    imprime(SecIs)
    println("}")
```

```
imprime(EOF()) :
    println(" < EOF > ")
```

4.1.1. Declaraciones

```
imprime(si_decs(LDecs)) :
    imprime(LDecs)
    println("&&")
```

```
imprime(no_decs()) : noop
```

```
imprime(muchas_decs(LDecs, Dec)) :
    imprime(LDecs)
    println("; ")
    imprime(Dec)
```

```
imprime(una_dec(Dec)) :
    imprime(Dec)
```

```
imprime(dec_base(TipoNom)) :
    imprime(TipoNom)
```

```
imprime(dec_type(TipoNom)) :
    println(" < type > ")
    imprime(TipoNom)
```

```
imprime(dec_proc(iden, ParamFs, Bloq)) :
    println(" < proc > ")
    println(iden)
    println("(")
    imprime(ParamFs)
    println(")")
    imprime(Bloq)
```

```
imprime(si_params_f(LParamFs)) :
    imprime(LParamFs)
```

```
imprime(no_params_f()) : noop
```

```
imprime(muchos_params_f(LPParamFs, ParamF)) :
    imprime(LPParamFs)
    println(", ")
    imprime(ParamF)
```

```
imprime(un_param_f(ParamF)) :
    imprime(ParamF)
```

```
imprime(si_refparam_f(Tipo, iden)) :
    imprime(Tipo)
    println("&")
    println(iden)
```

```
imprime(no_refparam_f(Tipo, iden)) :
    imprime(Tipo)
    println(iden)
```

4.1.2. Tipos

```
imprime(tipo_nombre(Tipo, iden)) :
    imprime(Tipo)
    println(iden)
```

```
imprime(tipo_array(Tipo, litEntero)) :
    imprime(Tipo)
    println("[")
    println(litEntero)
    println("]")
```

```
imprime(tipo_indir(Tipo)) :
    println("^")
    imprime(Tipo)
```

```
imprime(tipo_struct(LCampos)) :
    println("< struct > ")
    println("{")
    imprime(LCampos)
    println("}")
```

```
imprime(tipo_int()) :
    println("< int > ")
```

```
imprime(tipo_real()) :
    println("< real > ")
```

```
imprime(tipo_bool()) :
    println("< bool > ")
```

```
imprime(tipo_string()) :
    println("< string > ")
```

```
imprime(tipo_type(iden)) :
    println(iden)
```

```
imprime(muchos_campos(LCampos, TipoNom)) :
    imprime(LCampos)
    println(", ")
    imprime(TipoNom)
```

```
imprime(un_campo(TipoNom)) :
    imprime(TipoNom)
```

4.1.3. Instrucciones

```
imprime(si_ins(LIs)) :
    imprime(LIs)
```

```
imprime(no_ins()) : noop
```

```
imprime(muchas_ins(LIs, I)) :
    imprime(LIs)
    println(";")
    imprime(I)
```

```
imprime(una_ins(I)) :
    imprime(I)
```

```
imprime(ins_eval(Exp)) :
    println("< @ >")
    imprime(Exp)
```

```
imprime(ins_if(Exp, Bloq)) :
    println("< if >")
    imprime(Exp)
    imprime(Bloq)
```

```
imprime(ins_if_else(I, Bloq)) :
    imprime(I)
    println("< else >")
    imprime(Bloq)
```

```
imprime(ins_while(Exp, Bloq)) :
    println("< while >")
    imprime(Exp)
    imprime(Bloq)
```

```
imprime(ins_read(Exp)) :
    println("< read >")
    imprime(Exp)
```

```
imprime(ins_write(Exp)) :
    println("< write >")
    imprime(Exp)
```

```
imprime(ins_nl(Exp)) :
    println("< nl >")
```

```
imprime(ins_new(< Exp >)) :
    println("< new >")
    imprime(Exp)
```

```
imprime(ins_delete(Exp)) :
    println("< delete >")
    imprime(Exp)
```

```
imprime(ins_call(iden, ParamRs)) :
    println("< call >")
    println(iden)
    println("(")
```



```

    imprime(ParamRs)
    println("")

imprime(ins_bloque(Bloq)) :
    imprime(Bloq)

imprime(si_params_r(LParamRs)) :
    imprime(LParamRs)

imprime(no_params_r()) : noop

imprime(muchos_params_r(LParamRs, Exp)) :
    imprime(LParamRs)
    println(", ")
    imprime(Exp)

imprime(un_param_r(Exp)) :
    imprime(Exp)

```

4.1.4. Expresiones

```

imprime(exp_asig(Opnd0, Opnd1)) :
    imprimeExpBin(Opnd0, " = ", Opnd1, 1, 0)

imprime(exp_menor(Opnd0, Opnd1)) :
    imprimeExpBin(Opnd0, " < ", Opnd1, 1, 2)

imprime(exp_menor_ig(Opnd0, Opnd1)) :
    imprimeExpBin(Opnd0, " <= ", Opnd1, 1, 2)

imprime(exp_mayor(Opnd0, Opnd1)) :
    imprimeExpBin(Opnd0, " > ", Opnd1, 1, 2)

imprime(exp_mayor_ig(Opnd0, Opnd1)) :
    imprimeExpBin(Opnd0, " >= ", Opnd1, 1, 2)

imprime(exp_ig(Opnd0, Opnd1)) :
    imprimeExpBin(Opnd0, " == ", Opnd1, 1, 2)

imprime(exp_dist(Opnd0, Opnd1)) :
    imprimeExpBin(Opnd0, " != ", Opnd1, 1, 2)

imprime(exp_suma(Opnd0, Opnd1)) :
    imprimeExpBin(Opnd0, " + ", Opnd1, 2, 3)

imprime(exp_resta(Opnd0, Opnd1)) :
    imprimeExpBin(Opnd0, " - ", Opnd1, 3, 3)

imprime(exp_and(Opnd0, Opnd1)) :
    imprimeExpBin(Opnd0, " and ", Opnd1, 4, 3)

imprime(exp_or(Opnd0, Opnd1)) :
    imprimeExpBin(Opnd0, " or ", Opnd1, 4, 4)

imprime(exp_mul(Opnd0, Opnd1)) :
    imprimeExpBin(Opnd0, " * ", Opnd1, 4, 5)

imprime(exp_div(Opnd0, Opnd1)) :
    imprimeExpBin(Opnd0, " / ", Opnd1, 4, 5)

```

```

imprime(exp_mod(Opnd0, Opnd1)) :
    imprimeExpBin(Opnd0, " %", Opnd1, 4, 5)

imprime(exp_menos(Opnd)) :
    println(" - ")
    imprimeOpnd(Opnd, 5)

imprime(exp_not(Opnd)) :
    println(" not")
    imprimeOpnd(Opnd, 5)

imprime(exp_index(Opnd0, Opnd1)) :
    imprimeOpnd(Opnd0, 6)
    println(" [")
    imprimeOpnd(Opnd1, 0)
    println(" ]")

imprime(exp_reg(Opnd, iden)) :
    imprimeOpnd(Opnd, 6)
    println(" .")
    println(iden)

imprime(exp_indir(Opnd)) :
    imprimeOpnd(Opnd, 6)
    println(" ^")

imprime(exp_entero(litEntero)) :
    println(litEntero)
imprime(exp_real(litReal)) :
    println(litReal)

imprime(exp_true()) :
    println(" < true > ")

imprime(exp_false()) :
    println(" < false > ")

imprime(exp_cadena(litCadena)) :
    println(litCadena)

imprime(exp_iden(iden)) :
    println(iden)

imprime(exp_null()) :
    println(" < null > ")

imprimeExpBin(Opnd0, Op, Opnd1, np0, np1) :
    imprimeOpnd(Opnd0, np0)
    println(Op)
    imprimeOpnd(Opnd1, np1)

imprimeOpnd(Opnd, MinPrior) :
    if prioridad(Opnd) < MinPrior
        println(" (")
    end if
    imprime(Opnd)
    if prioridad(Opnd) < MinPrior
        println(" ")
    end if

```

```
prioridad(exp_asig(_, _)) : return 0
prioridad(exp_menor(_, _)) : return 1
prioridad(exp_menor_ig(_, _)) : return 1
prioridad(exp_mayor(_, _)) : return 1
prioridad(exp_mayor_ig(_, _)) : return 1
prioridad(exp_ig(_, _)) : return 1
prioridad(exp_dist(_, _)) : return 1
prioridad(exp_suma(_, _)) : return 2
prioridad(exp_resta(_, _)) : return 2
prioridad(exp_and(_, _)) : return 3
prioridad(exp_or(_, _)) : return 3
prioridad(exp_mul(_, _)) : return 4
prioridad(exp_div(_, _)) : return 4
prioridad(exp_mod(_, _)) : return 4
prioridad(exp_menos(_, _)) : return 5
prioridad(exp_not(_, _)) : return 5
prioridad(exp_index(_, _)) : return 6
prioridad(exp_reg(_, _)) : return 6
prioridad(exp_indir(_, _)) : return 6
```

4.1.5. Funciones auxiliares

```
println(str) :
    print str
    nl
```