

# Memoria: Procesadores de Lenguaje - Lenguaje Tiny

Fase 3: Desarrollo de constructores de ASTs

Grupo G03:

Burgos Sosa Rodrigo, Cassin Gina Andrea,  
Estebán Velasco Luis, Rabbia Santiago Elias

Curso 2024

## 1 Introducción

En el siguiente documento se expondrá una memoria sobre el desarrollo de constructores de ASTs descendentes y ascendentes sobre el lenguaje de programación Tiny. Se presentará una especificación abstracta del lenguaje mediante la enumeración de las firmas de las funciones constructoras de ASTs, se especificará el constructor de ASTs mediante una gramática s-atribuida y se acondicionará para permitir la implementación descendente. También se especificará un procesamiento para imprimir los tokens del programa leído.

## 2 Especificación de la sintaxis abstracta

A continuación se presentará la especificación de la sintaxis abstracta del lenguaje Tiny. Esto se hará siguiendo los siguientes pasos:

1. Eliminar terminales que no tienen carga semántica.
2. Fusionar todos los terminales que sean equivalentes entre sí.
3. Simplificar la gramática aplicando transformaciones.

- programa  $\rightarrow$  bloque
- bloque  $\rightarrow \{ \text{declaraciones instrucciones} \}$
- declaraciones  $\rightarrow$  declaracionesAux
- declaraciones  $\rightarrow \varepsilon$
- declaracionesAux  $\rightarrow$  declaracionesAux declaracion
- declaracionesAux  $\rightarrow$  declaracion
- declaracion  $\rightarrow$  tipo identificador
- declaracion  $\rightarrow$  type tipo identificador
- declaracion  $\rightarrow$  identificador paramsFormales bloque
- paramsFormales  $\rightarrow$  paramsFormalesLista
- paramsFormales  $\rightarrow \varepsilon$

- $\text{paramsFormalesLista} \rightarrow \text{paramsFormalesLista param}$
- $\text{paramsFormalesLista} \rightarrow \text{param}$
- $\text{param} \rightarrow \text{tipo \& identificador}$
- $\text{param} \rightarrow \text{tipo identificador}$
- $\text{tipo} \rightarrow \text{tipo literalEntero}$
- $\text{tipo} \rightarrow \text{\textasciicircum tipo}$
- $\text{tipo} \rightarrow \text{struct listaCampos}$
- $\text{tipo} \rightarrow \text{int}$
- $\text{tipo} \rightarrow \text{real}$
- $\text{tipo} \rightarrow \text{bool}$
- $\text{tipo} \rightarrow \text{string}$
- $\text{tipo} \rightarrow \text{identificador}$
- $\text{listaCampos} \rightarrow \text{listaCampos campo}$
- $\text{listaCampos} \rightarrow \text{campo}$
- $\text{campo} \rightarrow \text{tipo identificador}$
- $\text{instrucciones} \rightarrow \text{instruccionesAux}$
- $\text{instrucciones} \rightarrow \varepsilon$
- $\text{instruccionesAux} \rightarrow \text{instruccionesAux instruccion}$
- $\text{instruccionesAux} \rightarrow \text{instruccion}$
- $\text{instruccion} \rightarrow \text{E}$
- $\text{instruccion} \rightarrow \text{if E bloque}$
- $\text{instruccion} \rightarrow \text{if E bloque else bloque}$
- $\text{instruccion} \rightarrow \text{while E bloque}$
- $\text{instruccion} \rightarrow \text{read E}$

- $\text{instruccion} \rightarrow \text{write } E$
- $\text{instruccion} \rightarrow \text{nl}$
- $\text{instruccion} \rightarrow \text{new } E$
- $\text{instruccion} \rightarrow \text{delete } E$
- $\text{instruccion} \rightarrow \text{identificador paramsReales}$
- $\text{instruccion} \rightarrow \text{bloque}$
- $\text{paramsReales} \rightarrow \text{paramsRealesLista}$
- $\text{paramsReales} \rightarrow \varepsilon$
- $\text{paramsRealesLista} \rightarrow \text{paramsRealesLista } E$
- $\text{paramsRealesLista} \rightarrow E$
  
- $E \rightarrow E = E$
- $E \rightarrow E < E$
- $E \rightarrow E > E$
- $E \rightarrow E \leq E$
- $E \rightarrow E \geq E$
- $E \rightarrow E == E$
- $E \rightarrow E \neq E$
- $E \rightarrow E + E$
- $E \rightarrow E - E$
- $E \rightarrow E \text{ and } E$
- $E \rightarrow E \text{ or } E$
- $E \rightarrow E * E$
- $E \rightarrow E / E$
- $E \rightarrow E \% E$

- $E \rightarrow - E$
- $E \rightarrow \text{not } E$
- $E \rightarrow E [ E ]$
- $E \rightarrow E . \text{identificador}$
- $E \rightarrow E ^$
- $E \rightarrow \text{literalEntero}$
- $E \rightarrow \text{literalReal}$
- $E \rightarrow \text{true}$
- $E \rightarrow \text{false}$
- $E \rightarrow \text{literalCadena}$
- $E \rightarrow \text{identificador}$
- $E \rightarrow \text{null}$

A partir de lo generado anteriormente, se eligen nombres mnemotécnicos significativos para los géneros, especificado a continuación.

## 2.1 Géneros

NO TERMINAL	GÉNERO
programa	Prog
bloque	Bloq
declaraciones	Decs
declaracionesAux	DecsAux
declaracion	Dec
paramsFormales	ParamsF
paramsFormalesLista	ParamsFL
param	Param
tipo	T
listaCampos	LCampos
campo	Campo
instrucciones	Insts
instruccionesAux	InstsAux

instruccion	Inst
paramsReales	ParamsR
paramsRealesLista	ParamsRL
E	Exp

## 2.2 Constructores

A partir de los géneros contruidos arriba, se especifican los constructores.

REGLA	CONSTRUCTOR
programa $\rightarrow$ bloque	prog: Bloq $\rightarrow$ Prog
bloque $\rightarrow$ declaraciones instrucciones	bloq: Decs x Insts $\rightarrow$ Prog
declaraciones $\rightarrow$ declaracionesAux	si_dec: DecsAux $\rightarrow$ Decs
declaraciones $\rightarrow \varepsilon$	no_dec: $\rightarrow$ Decs
declaracionesAux $\rightarrow$ declaracionesAux declaracion	muchas_dec: DecsAux x Dec $\rightarrow$ DecsAux
declaracionesAux $\rightarrow$ declaracion	una_dec: Dec $\rightarrow$ DecsAux
declaracion $\rightarrow$ tipo identificador	dec_var: T x string $\rightarrow$ Dec
declaracion $\rightarrow$ type tipo identificador	dec_tipo: T x string $\rightarrow$ Dec
declaracion $\rightarrow$ identificador paramsFormales bloque	dec_proc: string x ParamsF x Bloq $\rightarrow$ Dec
paramsFormales $\rightarrow$ paramsFormalesLista	si_paramF: ParamsFL $\rightarrow$ ParamsF
paramsFormales $\rightarrow \varepsilon$	no_paramF: $\rightarrow$ ParamsF
paramsFormalesLista $\rightarrow$ paramsFormalesLista param	muchos_paramsF: ParamsFL x Param $\rightarrow$ ParamsFL
paramsFormalesLista $\rightarrow$ param	un_paramF: Param $\rightarrow$ ParamsFL
param $\rightarrow$ tipo & identificador	param_ref: T x string $\rightarrow$ Param
param $\rightarrow$ tipo identificador	param: T x string $\rightarrow$ Param
tipo $\rightarrow$ tipo literalEntero	tipo_array: T x string $\rightarrow$ T
tipo $\rightarrow \wedge$ tipo	tipo_punt: T $\rightarrow$ T
tipo $\rightarrow$ struct listaCampos	tipo_struct: LCampos $\rightarrow$ T
listaCampos $\rightarrow$ listaCampos campo	muchos_campos: LCampos x Campo $\rightarrow$ LCampos
listaCampos $\rightarrow$ campo	un_campo: Campo $\rightarrow$ LCampos
campo $\rightarrow$ tipo identificador	campo: T x string $\rightarrow$ Campo
tipo $\rightarrow$ int	tipo_int: $\rightarrow$ T
tipo $\rightarrow$ real	tipo_real: $\rightarrow$ T

tipo $\rightarrow$ bool	tipo_bool: $\rightarrow$ T
tipo $\rightarrow$ string	tipo_string: $\rightarrow$ T
tipo $\rightarrow$ identificador	tipo_iden: string $\rightarrow$ T
instrucciones $\rightarrow$ instruccionesAux	si_instr: InstAux $\rightarrow$ Insts
instrucciones $\rightarrow \varepsilon$	no_instr: $\rightarrow$ Insts
instruccionesAux $\rightarrow$ instruccionesAux instruccion	muchas_instr: InstsAux x Inst $\rightarrow$ InstsAux
instruccionesAux $\rightarrow$ instruccion	una_instr: Inst $\rightarrow$ InstsAux
instruccion $\rightarrow$ E	instr_expr: Exp $\rightarrow$ Inst
instruccion $\rightarrow$ if E bloque	instr_if: Exp x Bloq $\rightarrow$ Inst
instruccion $\rightarrow$ if E bloque else bloque	instr_if_else: Exp x Bloq x Bloq $\rightarrow$ Inst
instruccion $\rightarrow$ while E bloque	instr_while: Exp x Bloq $\rightarrow$ Inst
instruccion $\rightarrow$ read E	instr_read: Exp $\rightarrow$ Inst
instruccion $\rightarrow$ write E	instr_write: Exp $\rightarrow$ Inst
instruccion $\rightarrow$ nl	instr_nl: $\rightarrow$ Inst
instruccion $\rightarrow$ new E	instr_new: Exp $\rightarrow$ Inst
instruccion $\rightarrow$ delete E	instr_del: Exp $\rightarrow$ Inst
instruccion $\rightarrow$ identificador paramsReales	instr_call: string x ParamsR $\rightarrow$ Inst
instruccion $\rightarrow$ bloque	instr_bloque: Bloq $\rightarrow$ Inst
paramsReales $\rightarrow$ paramsRealesLista	si_paramsR: ParamsRL $\rightarrow$ ParamsR
paramsReales $\rightarrow \varepsilon$	no_paramsR: $\rightarrow$ ParamsR
paramsRealesLista $\rightarrow$ paramsRealesLista E	muchos_paramsR: ParamsRL x Exp $\rightarrow$ ParamsRL
paramsRealesLista $\rightarrow$ E	un_paramsR: Exp $\rightarrow$ ParamsRL
E $\rightarrow$ E = E	asig: Exp x Exp $\rightarrow$ Exp
E $\rightarrow$ E i E	menor: Exp x Exp $\rightarrow$ Exp
E $\rightarrow$ E l E	mayor: Exp x Exp $\rightarrow$ Exp
E $\rightarrow$ E i= E	menor_igual: Exp x Exp $\rightarrow$ Exp
E $\rightarrow$ E l= E	mayor_igual: Exp x Exp $\rightarrow$ Exp
E $\rightarrow$ E == E	igual: Exp x Exp $\rightarrow$ Exp
E $\rightarrow$ E != E	no_igual: Exp x Exp $\rightarrow$ Exp
E $\rightarrow$ E + E	suma: Exp x Exp $\rightarrow$ Exp
E $\rightarrow$ E - E	resta: Exp x Exp $\rightarrow$ Exp
E $\rightarrow$ E and E	and: Exp x Exp $\rightarrow$ Exp
E $\rightarrow$ E or E	or: Exp x Exp $\rightarrow$ Exp
E $\rightarrow$ E * E	mult: Exp x Exp $\rightarrow$ Exp
E $\rightarrow$ E / E	div: Exp x Exp $\rightarrow$ Exp

$E \rightarrow E \% E$	mod: $\text{Exp} \times \text{Exp} \rightarrow \text{Exp}$
$E \rightarrow - E$	negativo: $\text{Exp} \rightarrow \text{Exp}$
$E \rightarrow \text{not } E$	not: $\text{Exp} \rightarrow \text{Exp}$
$E \rightarrow E [E]$	index: $\text{Exp} \times \text{Exp} \rightarrow \text{Exp}$
$E \rightarrow E . \text{identificador}$	acceso: $\text{Exp} \times \text{string} \rightarrow \text{Exp}$
$E \rightarrow E ^$	indireccion: $\text{Exp} \rightarrow \text{Exp}$
$E \rightarrow \text{literalEntero}$	lit_ent: $\text{string} \rightarrow \text{Exp}$
$E \rightarrow \text{literaReal}$	lit_real: $\text{string} \rightarrow \text{Exp}$
$E \rightarrow \text{true}$	true: $\rightarrow \text{Exp}$
$E \rightarrow \text{false}$	false: $\rightarrow \text{Exp}$
$E \rightarrow \text{literalCadena}$	lit_cadena: $\text{string} \rightarrow \text{Exp}$
$E \rightarrow \text{identificador}$	iden: $\text{string} \rightarrow \text{Exp}$
$E \rightarrow \text{null}$	null: $\rightarrow \text{Exp}$

### 3 Especificación del constructor de ASTs mediante una gramática s-atribuida

A partir de lo obtenido previamente, podemos especificar el constructor de ASTs con una gramática s-atribuida.

- $\text{programa} \rightarrow \text{bloque}$   
 $\text{programa.a} = \text{prog}(\text{bloque a})$
- $\text{bloque} \rightarrow \{ \text{declaraciones instrucciones} \}$   
 $\text{bloque.a} = \text{bloq}(\text{declaraciones.a}, \text{instrucciones.a})$
- $\text{declaraciones} \rightarrow \text{declaracionesAux} \ \&\&$   
 $\text{declaraciones.a} = \text{si\_decs}(\text{declaracionesAux.a})$
- $\text{declaraciones} \rightarrow \varepsilon$   
 $\text{declaraciones.a} = \text{no\_decs}()$
- $\text{declaracionesAux} \rightarrow \text{declaracionesAux} ; \text{declaracion}$   
 $\text{declaracionesAux1.a} = \text{muchas\_decs}(\text{declaracionesAux2.a}, \text{declaracion.a})$



- $\text{declaracionesAux} \rightarrow \text{declaracion}$   
 $\text{declaracionesAux.a} = \text{una\_dec}(\text{declaracion.a})$
- $\text{declaracion} \rightarrow \text{declaracionVar}$   
 $\text{declaracion.a} = \text{declaracionVar.a}$
- $\text{declaracion} \rightarrow \text{declaracionTipo}$   
 $\text{declaracion} = \text{declaracionTipo.a}$
- $\text{declaracion} \rightarrow \text{declaracionProc}$   
 $\text{declaracion} = \text{declaracionProc.a}$
- $\text{declaracionVar} \rightarrow \text{tipo0 identificador}$   
 $\text{declaracionVar.a} = \text{dec\_var}(\text{tipo0.a}, \text{identificador.lex})$
- $\text{declaracionTipo} \rightarrow \text{type tipo0 identificador}$   
 $\text{declaracionTipo.a} = \text{dec\_tipo}(\text{tipo0.a}, \text{identificador.lex})$
- $\text{declaracionProc} \rightarrow \text{proc identificador paramsFormales bloque}$   
 $\text{declaracionProc.a} = \text{dec\_proc}(\text{identificador.lex}, \text{paramsFormales.a}, \text{bloque.a})$
- $\text{paramsFormales} \rightarrow ( \text{paramsFormalesAux} )$   
 $\text{paramsFormales.a} = \text{paramsFormalesAux.a}$
- $\text{paramsFormalesAux} \rightarrow \text{paramsFormalesLista}$   
 $\text{paramsFormalesAux.a} = \text{si\_params}(\text{paramsFormalesLista.a})$
- $\text{paramsFormalesAux} \rightarrow \varepsilon$   
 $\text{paramsFormalesAux.a} = \text{no\_params}()$
- $\text{paramsFormalesLista} \rightarrow \text{paramsFormalesLista} , \text{param}$   
 $\text{paramsFormalesLista.a} = \text{muchos\_params}(\text{paramsFormalesLista.a}, \text{param.a})$
- $\text{paramsFormalesLista} \rightarrow \text{param}$   
 $\text{paramsFormalesLista.a} = \text{un\_param}(\text{param.a})$
- $\text{param} \rightarrow \text{tipo0 \& identificador}$   
 $\text{param.a} = \text{param\_ref}(\text{tipo0.a}, \text{identificador.lex})$

- $\text{param} \rightarrow \text{tipo0 identificador}$   
 $\text{param.a} = \text{param}(\text{tipo0.a}, \text{identificador.lex})$
- $\text{tipo0} \rightarrow \text{tipo0 literalEntero}$   
 $\text{tipo0}_1.\text{a} = \text{tipo\_array}(\text{tipo0}_2.\text{a}, \text{literalEntero.lex})$
- $\text{tipo0} \rightarrow \text{tipo1}$   
 $\text{tipo0.a} = \text{tipo.a}$
- $\text{tipo1} \rightarrow \wedge \text{tipo1}$   
 $\text{tipo1}_1.\text{a} = \text{tipo\_punt}(\text{tipo1}_2.\text{a})$
- $\text{tipo1} \rightarrow \text{tipo2}$   
 $\text{tipo1.a} = \text{tipo2.a}$
- $\text{tipo2} \rightarrow \text{struct } \{ \text{listaCampos} \}$   
 $\text{tipo2.a} = \text{tipo\_struct}(\text{listaCampos.a})$
- $\text{listaCampos} \rightarrow \text{listaCampos}, \text{campo}$   
 $\text{listaCampos1.a} = \text{muchos\_campos}(\text{listaCampos2.a}, \text{campo.a})$
- $\text{listaCampos} \rightarrow \text{campo}$   
 $\text{listaCampos.a} = \text{un\_campo}(\text{campo.a})$
- $\text{campo} \rightarrow \text{tipo0 identificador}$   
 $\text{campo.a} = \text{campo}(\text{tipo0.a}, \text{identificador.lex})$
- $\text{tipo2} \rightarrow \text{int}$   
 $\text{tipo2.a} = \text{tipo\_int}()$
- $\text{tipo2} \rightarrow \text{real}$   
 $\text{tipo2.a} = \text{tipo\_real}()$
- $\text{tipo2} \rightarrow \text{bool}$   
 $\text{tipo2.a} = \text{tipo\_bool}()$
- $\text{tipo2} \rightarrow \text{string}$   
 $\text{tipo2.a} = \text{tipo\_string}()$

- $\text{tipo2} \rightarrow \text{identificador}$   
 $\text{tipo2.a} = \text{tipo\_iden}(\text{string.lex})$
- $\text{instrucciones} \rightarrow \text{instruccionesAux}$   
 $\text{instrucciones.a} = \text{si\_instr}(\text{instruccionesAux.a})$
- $\text{instrucciones} \rightarrow \varepsilon$   
 $\text{instrucciones.a} = \text{no\_instr}()$
- $\text{instruccionesAux} \rightarrow \text{instruccionesAux} ; \text{instruccion}$   
 $\text{instruccionesAux1.a} = \text{muchas\_instr}(\text{instruccionesAux2.a}, \text{instruccion.a})$
- $\text{instruccionesAux} \rightarrow \text{instruccion}$   
 $\text{instruccionesAux.a} = \text{una\_instr}(\text{instruccion.a})$
- $\text{instruccion} \rightarrow @ \text{expr}$   
 $\text{instruccion.a} = \text{instr\_expr}(\text{expr.a})$
- $\text{instruccion} \rightarrow \text{if expr bloque}$   
 $\text{instruccion.a} = \text{instr\_if}(\text{expr.a}, \text{bloque.a})$
- $\text{instruccion} \rightarrow \text{if expr bloque else bloque}$   
 $\text{instruccion.a} = \text{instr\_if\_else}(\text{expr.a}, \text{bloque1.a}, \text{bloque2.a})$
- $\text{instruccion} \rightarrow \text{while expr bloque}$   
 $\text{instruccion.a} = \text{instr\_while}(\text{expr.a}, \text{bloque.a})$
- $\text{instruccion} \rightarrow \text{read expr}$   
 $\text{instruccion.a} = \text{instr\_read}(\text{expr.a})$
- $\text{instruccion} \rightarrow \text{write expr}$   
 $\text{instruccion.a} = \text{instr\_write}(\text{expr.a})$
- $\text{instruccion} \rightarrow \text{nl}$   
 $\text{instruccion.a} = \text{instr\_nl}()$
- $\text{instruccion} \rightarrow \text{new expr}$   
 $\text{instruccion.a} = \text{instr\_new}(\text{expr.a})$

- $\text{instruccion} \rightarrow \text{delete expr}$   
 $\text{instruccion.a} = \text{instr\_del}(\text{expr.a})$
- $\text{instruccion} \rightarrow \text{call identificador paramsReales}$   
 $\text{instruccion.a} = \text{instr\_call}(\text{identificador.lex}, \text{paramsReales.a})$
- $\text{instruccion} \rightarrow \text{bloque}$   
 $\text{instruccion.a} = \text{instr\_bloque}(\text{bloque.a})$
- $\text{paramsReales} \rightarrow ( \text{paramsRealesAux} )$   
 $\text{paramsReales.a} = \text{paramsRealesAux.a}$
- $\text{paramsRealesAux} \rightarrow \text{paramsRealesLista}$   
 $\text{paramsRealesAux.a} = \text{si\_paramsR}(\text{paramsRealesLista.a})$
- $\text{paramsRealesAux} \rightarrow \varepsilon$   
 $\text{paramsRealesAux.a} = \text{no\_paramsR}()$
- $\text{paramsRealesLista} \rightarrow \text{paramsRealesLista}, \text{expr}$   
 $\text{paramsRealesLista1.a} = \text{muchos\_paramsR}(\text{paramsRealesLista2.a}, \text{expr.a})$
- $\text{paramsRealesLista} \rightarrow \text{expr}$   
 $\text{paramsRealesLista.a} = \text{un\_paramsR}(\text{expr.a})$
- $\text{expr} \rightarrow e0$   
 $\text{expr.a} = e0.a$
- $e0 \rightarrow e1 = e0$   
 $e0_1.a = \text{asig}(e1.a, e0_2.a)$
- $e0 \rightarrow e1$   
 $e0.a = e1.a$
- $e1 \rightarrow e1 \text{ op1 } e2$   
 $e1_1.a = \text{mkop1}(\text{op1.op}, e1_2.a, e2.a)$
- $e1 \rightarrow e2$   
 $e1.a = e2.a$

- $e2 \rightarrow e2 + e3$   
 $e2_1.a = \text{suma}(e2_2.a, e3.a)$
- $e2 \rightarrow e3 - e3$   
 $e2.a = \text{resta}(e3_1.a, e3_2.a)$
- $e2 \rightarrow e3$   
 $e2.a = e3.a$
- $e3 \rightarrow e4 \text{ and } e3$   
 $e3_1.a = \text{and}(e4.a, e3_2.a)$
- $e3 \rightarrow e4 \text{ or } e4$   
 $e3.a = \text{or}(e4_1.a, e4_2.a)$
- $e3 \rightarrow e4$   
 $e3.a = e4.a$
- $e4 \rightarrow e4 \text{ op4 } e5$   
 $e4_1.a = \text{mkop4}(\text{op4.op}, e4_2.a, e5.a)$
- $e4 \rightarrow e5$   
 $e4.a = e5.a$
- $e5 \rightarrow \text{op5 } e5$   
 $e5_1.a = \text{mkop5}(\text{op5.op}, e5_2.a)$
- $e5 \rightarrow e6$   
 $e5.a = e6.a$
- $e6 \rightarrow e6 [ \text{expr} ]$   
 $e6_1.a = \text{index}(e6_2.a, \text{expr}.a)$
- $e6 \rightarrow e6 . \text{identificador}$   
 $e6_1.a = \text{acceso}(e6_2.a, \text{identificador.lex})$
- $e6 \rightarrow e6 ^$   
 $e6_1.a = \text{indireccion}(e6_2.a)$

- $e6 \rightarrow e7$   
 $e6.a = e7.a$
- $e7 \rightarrow ( e0 )$   
 $e7.a = e0.a$
- $e7 \rightarrow \text{literalEntero}$   
 $e7.a = \text{lit\_ent}(\text{literalEntero.lex})$
- $e7 \rightarrow \text{literaReal}$   
 $e7.a = \text{lit\_Real}(\text{literalReal.lex})$
- $e7 \rightarrow \text{true}$   
 $e7.a = \text{lit\_true}()$
- $e7 \rightarrow \text{false}$   
 $e7.a = \text{lit\_false}()$
- $e7 \rightarrow \text{literalCadena}$   
 $e7.a = \text{lit\_cadena}(\text{literalCadena.lex})$
- $e7 \rightarrow \text{identificador}$   
 $e7.a = \text{identificador.lex}$
- $e7 \rightarrow \text{null}$   
 $e7.a = \text{null}()$
- $op1 \rightarrow <$   
 $op1.op = "<"$
- $op1 \rightarrow >$   
 $op1.op = ">"$
- $op1 \rightarrow <=$   
 $op1.op = "<="$
- $op1 \rightarrow >=$   
 $op1.op = ">="$

- $\text{op1} \rightarrow ==$   
 $\text{op1.op} = “==”$
- $\text{op1} \rightarrow !=$   
 $\text{op1.op} = “!=”$
- $\text{op4} \rightarrow *$   
 $\text{op4.op} = “*”$
- $\text{op4} \rightarrow /$   
 $\text{op4.op} = “/”$
- $\text{op4} \rightarrow \%$   
 $\text{op4.op} = “\%”$
- $\text{op5} \rightarrow -$   
 $\text{op5.op} = “-”$
- $\text{op5} \rightarrow \text{not}$   
 $\text{op5.op} = “\text{not}”$
- $\text{fun mkop1}(\text{op}, \text{opnd1}, \text{opnd2})$ :  
 $\text{op} = “<” \rightarrow \text{return menor}(\text{opnd1}, \text{opnd2})$   
 $\text{op} = “>” \rightarrow \text{return mayor}(\text{opnd1}, \text{opnd2})$   
 $\text{op} = “<=” \rightarrow \text{return menor\_igual}(\text{opnd1}, \text{opnd2})$   
 $\text{op} = “>=” \rightarrow \text{return mayor\_igual}(\text{opnd1}, \text{opnd2})$   
 $\text{op} = “==” \rightarrow \text{return igual}(\text{opnd1}, \text{opnd2})$   
 $\text{op} = “!=” \rightarrow \text{return no\_igual}(\text{opnd1}, \text{opnd2})$
- $\text{fun mkop4}(\text{op}, \text{opnd1}, \text{opnd2})$ :  
 $\text{op} = “*” \rightarrow \text{return mul}(\text{opnd1}, \text{opnd2})$   
 $\text{op} = “/” \rightarrow \text{return div}(\text{opnd1}, \text{opnd2})$   
 $\text{op} = “\%” \rightarrow \text{return mod}(\text{opnd1}, \text{opnd2})$
- $\text{fun mkop5}(\text{op}, \text{opnd})$ :  
 $\text{op} = “-” \rightarrow \text{return neg}(\text{opnd})$   
 $\text{op} = “\text{not}” \rightarrow \text{return not}(\text{opnd})$

## 4 Acondicionamiento para permitir la implementación descendente

- $\text{programa} \rightarrow \text{bloque}$   
 $\text{programa.a} = \text{prog}(\text{bloque.a})$
- $\text{bloque} \rightarrow \{ \text{declaraciones instrucciones} \}$   
 $\text{bloque.a} = \text{bloq}(\text{declaraciones.a}, \text{instrucciones.a})$
- $\text{declaraciones} \rightarrow \text{declaracionesAux} \ \&\&$   
 $\text{declaraciones.a} = \text{si\_decs}(\text{declaracionesAux.a})$
- $\text{declaraciones} \rightarrow \varepsilon$   
 $\text{declaraciones.a} = \text{no\_decs}()$
- $\text{declaracionesAux} \rightarrow \text{declaracion} \ \text{recDeclaracion}$   
 $\text{recDeclaracion.h} = \text{una\_dec}(\text{declaracion.a})$   
 $\text{declaracionesAux.a} = \text{recDeclaracion.a}$
- $\text{recDeclaracion} \rightarrow ; \text{declaracion} \ \text{recDeclaracion}$   
 $\text{recDeclaracion2.h} = \text{muchas\_decs}(\text{recDeclaracion1.h}, \text{declaraciones.a})$   
 $\text{recDeclaracion1.a} = \text{recDeclaracion2.a}$
- $\text{recDeclaracion} \rightarrow \varepsilon$   
 $\text{recDeclaracion.a} = \text{recDeclaracion.h}$
- $\text{declaracion} \rightarrow \text{declaracionVar}$   
 $\text{declaracion.a} = \text{declaracionVar.a}$
- $\text{declaracion} \rightarrow \text{declaracionTipo}$   
 $\text{declaracion} = \text{declaracionTipo.a}$
- $\text{declaracion} \rightarrow \text{declaracionProc}$   
 $\text{declaracion} = \text{declaracionProc.a}$
- $\text{declaracionVar} \rightarrow \text{tipo0} \ \text{identificador}$   
 $\text{declaracionVar.a} = \text{dec\_var}(\text{tipo0.a}, \text{identificador.lex})$



- $\text{declaracionTipo} \rightarrow \text{type tipo0 identificador}$   
 $\text{declaracionTipo.a} = \text{dec\_tipo}(\text{tipo0.a}, \text{identificador.lex})$
- $\text{declaracionProc} \rightarrow \text{proc identificador paramsFormales bloque}$   
 $\text{declaracionProc.a} = \text{dec\_proc}(\text{identificador.lex}, \text{paramsFormales.a}, \text{bloque.a})$
- $\text{paramsFormales} \rightarrow ( \text{paramsFormalesAux} )$   
 $\text{paramsFormales.a} = \text{paramsFormalesAux.a}$
- $\text{paramsFormalesAux} \rightarrow \text{paramsFormalesLista}$   
 $\text{paramsFormalesAux.a} = \text{si\_params}(\text{paramsFormalesLista.a})$
- $\text{paramsFormalesAux} \rightarrow \varepsilon$   
 $\text{paramsFormalesAux.a} = \text{no\_params}()$
- $\text{paramsFormalesLista} \rightarrow \text{param recParamFormal}$   
 $\text{recParamFormal.h} = \text{un\_param}(\text{param.a})$   
 $\text{paramsFormalesLista.a} = \text{recParamFormal.a}$
- $\text{recParamFormal} \rightarrow , \text{param recParamFormal}$   
 $\text{recParamFormal2.h} = \text{muchos\_params}(\text{recParamFormal1.h}, \text{param.a})$   
 $\text{recParamFormal1.a} = \text{recParamFormal2.a}$
- $\text{recParamFormal} \rightarrow \varepsilon$   
 $\text{recParamFormal.a} = \text{recParamFormal.h}$
- $\text{param} \rightarrow \text{tipo0 referencia identificador}$   
 $\text{referencia.ht} = \text{tipo0.a}$   
 $\text{referencia.hi} = \text{identificador.lex}$   
 $\text{param.a} = \text{referencia.a}$
- $\text{referencia} \rightarrow \&$   
 $\text{referencia.a} = \text{param\_ref}(\text{referencia.ht}, \text{referencia.hi})$
- $\text{referencia} \rightarrow \varepsilon$   
 $\text{referencia.a} = \text{param}(\text{referencia.ht}, \text{referencia.hi})$

- $\text{tipo0} \rightarrow \text{tipo1} \text{ recArray}$   
 $\text{recArray.h} = \text{tipo1.a}$   
 $\text{tipo0.a} = \text{recArray.a}$
- $\text{recArray} \rightarrow [ \text{literalEntero} ] \text{ recArray}$   
 $\text{recArray2.h} = \text{tipo\_array}(\text{recArray1.h}, \text{literalEntero.lex})$   
 $\text{recArray1.a} = \text{recArray2.a}$
- $\text{recArray} \rightarrow \varepsilon$   
 $\text{recArray.a} = \text{recArray.h}$
- $\text{tipo1} \rightarrow \wedge \text{ tipo1}$   
 $\text{tipo1}_1.\text{a} = \text{tipo\_punt}(\text{tipo1}_2.\text{a})$
- $\text{tipo1} \rightarrow \text{tipo2}$   
 $\text{tipo1.a} = \text{tipo2.a}$
- $\text{tipo2} \rightarrow \text{struct } \{ \text{listaCampos} \}$   
 $\text{tipo2.a} = \text{tipo\_struct}(\text{listaCampos.a})$
- $\text{listaCampos} \rightarrow \text{campo} \text{ recCampo}$   
 $\text{recCampo.h} = \text{un\_campo}(\text{campo.a})$   
 $\text{listaCampos.a} = \text{recCampo.a}$
- $\text{recCampo} \rightarrow , \text{ campo} \text{ recCampo}$   
 $\text{recCampo2.h} = \text{muchos\_campos}(\text{recCampo1.h}, \text{campo.a})$   
 $\text{recCampo1.a} = \text{recCampo2.a}$
- $\text{recCampo} \rightarrow \varepsilon$   
 $\text{recCampo.a} = \text{recCampo.h}$
- $\text{campo} \rightarrow \text{tipo0} \text{ identificador}$   
 $\text{campo.a} = \text{campo}(\text{tipo0.a}, \text{identificador.lex})$
- $\text{tipo2} \rightarrow \text{int}$   
 $\text{tipo2.a} = \text{tipo\_int}()$
- $\text{tipo2} \rightarrow \text{real}$   
 $\text{tipo2.a} = \text{tipo\_real}()$

- $\text{tipo2} \rightarrow \text{bool}$   
 $\text{tipo2.a} = \text{tipo\_bool}()$
- $\text{tipo2} \rightarrow \text{string}$   
 $\text{tipo2.a} = \text{tipo\_string}()$
- $\text{tipo2} \rightarrow \text{identificador}$   
 $\text{tipo2.a} = \text{tipo\_iden}(\text{string.lex})$
- $\text{instrucciones} \rightarrow \text{instruccionesAux}$   
 $\text{instrucciones.a} = \text{si\_instr}(\text{instruccionesAux.a})$
- $\text{instrucciones} \rightarrow \varepsilon$   
 $\text{instrucciones.a} = \text{no\_instr}()$
- $\text{instruccionesAux} \rightarrow \text{instruccion recInstruccion}$   
 $\text{recInstruccion.h} = \text{una\_instr}(\text{instruccion.a})$   
 $\text{instruccionesAux.a} = \text{recInstruccion.a}$
- $\text{recInstruccion} \rightarrow ; \text{instruccion recInstruccion}$   
 $\text{recInstruccion2.h} = \text{muchas\_instr}(\text{recInstruccion1.h}, \text{instruccion.a})$   
 $\text{recInstruccion1.a} = \text{recInstruccion2.a}$
- $\text{recInstruccion} \rightarrow \varepsilon$   
 $\text{recInstruccion.a} = \text{recInstruccion.h}$
- $\text{instruccion} \rightarrow @ \text{expr}$   
 $\text{instruccion.a} = \text{instr\_expr}(\text{expr.a})$
- $\text{instruccion} \rightarrow \text{if expr bloque facIf}$   
 $\text{facIf.h1} = \text{expr.a}$   
 $\text{facIf.h2} = \text{bloque.a}$   
 $\text{instruccion.a} = \text{factIf.a}$
- $\text{facIf} \rightarrow \text{else bloque}$   
 $\text{facIf.a} = \text{instr\_if\_else}(\text{facIf.h1}, \text{facIf.h2}, \text{bloque.a})$

- $\text{facIf} \rightarrow \varepsilon$   
 $\text{facIf.a} = \text{instr\_if}(\text{facIf.h1}, \text{facIf.h2})$
- $\text{instruccion} \rightarrow \text{while expr bloque}$   
 $\text{instruccion.a} = \text{instr\_while}(\text{expr.a}, \text{bloque.a})$
- $\text{instruccion} \rightarrow \text{read expr}$   
 $\text{instruccion.a} = \text{instr\_read}(\text{expr.a})$
- $\text{instruccion} \rightarrow \text{write expr}$   
 $\text{instruccion.a} = \text{instr\_write}(\text{expr.a})$
- $\text{instruccion} \rightarrow \text{nl}$   
 $\text{instruccion.a} = \text{instr\_nl}()$
- $\text{instruccion} \rightarrow \text{new expr}$   
 $\text{instruccion.a} = \text{instr\_new}(\text{expr.a})$
- $\text{instruccion} \rightarrow \text{delete expr}$   
 $\text{instruccion.a} = \text{instr\_del}(\text{expr.a})$
- $\text{instruccion} \rightarrow \text{call identificador paramsReales}$   
 $\text{instruccion.a} = \text{instr\_call}(\text{identificador.lex}, \text{paramsReales.a})$
- $\text{instruccion} \rightarrow \text{bloque}$   
 $\text{instruccion.a} = \text{instr\_bloque}(\text{bloque.a})$
- $\text{paramsReales} \rightarrow ( \text{paramsRealesAux} )$   
 $\text{paramsReales.a} = \text{paramsRealesAux.a}$
- $\text{paramsRealesAux} \rightarrow \text{paramsRealesLista}$   
 $\text{paramsRealesAux.a} = \text{si\_paramsR}(\text{paramsRealesLista.a})$
- $\text{paramsRealesAux} \rightarrow \varepsilon$   
 $\text{paramsRealesAux.a} = \text{no\_paramsR}()$
- $\text{paramsRealesLista} \rightarrow \text{expr recParamReal}$   
 $\text{recParamReal.h} = \text{un\_paramsR}(\text{expr.a})$   
 $\text{paramsRealesLista.a} = \text{recParamReal.a}$

- $\text{recParamReal} \rightarrow , \text{expr } \text{recParamReal}$ 

$$\text{recParamReal2.h} = \text{muchos\_paramsR}(\text{recParamReal1.h}, \text{expr.a})$$

$$\text{recParamReal1.a} = \text{recParamReal2.a}$$
- $\text{recParamReal} \rightarrow \varepsilon$ 

$$\text{recParamReal.a} = \text{recParamReal.h}$$
- $\text{expr} \rightarrow \text{e0}$
- $\text{e0} \rightarrow \text{e1 } \text{facE1}$ 

$$\text{facE1.h} = \text{e1.a}$$

$$\text{e0.a} = \text{facE1.a}$$
- $\text{facE1} \rightarrow = \text{e0 } \text{facE1}$ 

$$\text{facE1}_2.\text{h} = \text{asig}(\text{facE1}_1.\text{h}, \text{e0.a})$$

$$\text{facE1}_1.\text{a} = \text{facE1}_2.\text{a}$$
- $\text{facE1} \rightarrow \varepsilon$ 

$$\text{facE1.a} = \text{facE1.h}$$
- $\text{e1} \rightarrow \text{e2 } \text{recOp1}$ 

$$\text{recOp1.h} = \text{e2.a}$$

$$\text{e1.a} = \text{recOp1.a}$$
- $\text{recOp1} \rightarrow \text{op1 } \text{e2 } \text{recOp1}$ 

$$\text{recOp1}_2.\text{h} = \text{mkop1}(\text{op1.op}, \text{recOp1}_1.\text{h}, \text{e2.a})$$

$$\text{recOp1}_1.\text{a} = \text{recOp1}_2.\text{a}$$
- $\text{recOp1} \rightarrow \varepsilon$ 

$$\text{recOp1.a} = \text{recOp1.h}$$
- $\text{e2} \rightarrow \text{e3 } \text{facE3 } \text{recSuma}$ 

$$\text{facE3.h} = \text{e3.a}$$

$$\text{recSuma.h} = \text{facE3.a}$$

$$\text{e2.a} = \text{recSuma.a}$$

- $\text{recSuma} \rightarrow + \text{e3 recSuma}$   
 $\text{recSuma2.h} = \text{suma}(\text{recSuma1.h}, \text{e3.a})$   
 $\text{recSuma1.a} = \text{recSuma2.a}$
- $\text{recSuma} \rightarrow \varepsilon$   
 $\text{recSuma.a} = \text{recSuma.h}$
- $\text{facE3} \rightarrow - \text{e3}$   
 $\text{facE3.a} = \text{resta}(\text{fecE3.h}, \text{e3.a})$
- $\text{facE3} \rightarrow \varepsilon$   
 $\text{facE3.a} = \text{facE3.h}$
- $\text{e3} \rightarrow \text{e4 facE4}$   
 $\text{facE4.h} = \text{e4.a}$   
 $\text{e3.a} = \text{facE4.a}$
- $\text{facE4} \rightarrow \text{and e3}$   
 $\text{facE4.a} = \text{and}(\text{facE4.h}, \text{e3.a})$
- $\text{facE4} \rightarrow \text{or e4}$   
 $\text{facE4.a} = \text{or}(\text{facE4.h}, \text{e3.a})$
- $\text{facE4} \rightarrow \varepsilon$   
 $\text{facE4.a} = \text{facE4.h}$
- $\text{e4} \rightarrow \text{e5 recOp4}$   
 $\text{recOp4.h} = \text{e5.a}$   
 $\text{r4.a} = \text{recOp4.a}$
- $\text{recOp4} \rightarrow \text{op4 e5 recOp4}$   
 $\text{recOp4}_2.\text{h} = \text{mkop4}(\text{op4.op}, \text{recOp4}_1.\text{h}, \text{e5.a})$   
 $\text{recOp4}_1.\text{a} = \text{recOp4}_2.\text{a}$
- $\text{recOp4} \rightarrow \varepsilon$   
 $\text{recOp4.a} = \text{recOp4.h}$
- $\text{e5} \rightarrow \text{op5 e5}$   
 $\text{e5}_1.\text{a} = \text{mkop5}(\text{op5.op}, \text{e5}_2.\text{a})$

- $e5 \rightarrow e6$   
 $e5.a = e6.a$
- $e6 \rightarrow e7 \text{ recOp6}$   
 $\text{recOp6.h} = e6Aux.a$   
 $e6.a = \text{recOp6.a}$
- $\text{recOp6} \rightarrow [ \text{expr} ] \text{ recOp6}$   
 $\text{recOp6}_2.h = \text{index}(\text{recOp6.h}, \text{expr.a})$   
 $\text{recOp6}_1.a = \text{recOp6}_2.a$
- $\text{recOp6} \rightarrow . \text{ identificador recOp6}$   
 $\text{recOp6}_2.h = \text{acceso}(\text{recOp6.a}, \text{identificador.lex})$   
 $\text{recOp6}_1.a = \text{recOp6}_2.a$
- $\text{recOp6} \rightarrow ^ \text{ recOp6}$   
 $\text{recOp6}_2.h = \text{indireccion}(\text{recOp6.h})$   
 $\text{recOp6}_1.a = \text{recOp6}_2.a$
- $\text{recOp6} \rightarrow \varepsilon$   
 $\text{recOp6.a} = \text{recOp6.h}$
- $e7 \rightarrow ( e0 )$   
 $e7.a = e0.a$
- $e7 \rightarrow \text{literalEntero}$   
 $e7.a = \text{lit\_ent}(\text{literalEntero.lex})$
- $e7 \rightarrow \text{literaReal}$   
 $e7.a = \text{lit\_real}(\text{literalReal.lex})$
- $e7 \rightarrow \text{true}$   
 $e7.a = \text{lit\_true}()$
- $e7 \rightarrow \text{false}$   
 $e7.a = \text{lit\_false}()$
- $e7 \rightarrow \text{literalCadena}$   
 $e7.a = \text{lit\_cadena}(\text{literalCadena.lex})$

- $e7 \rightarrow \text{identificador}$   
 $e7.a = \text{identificador.lex}$
- $e7 \rightarrow \text{null}$   
 $e7.a = \text{null}()$
- $op1 \rightarrow <$   
 $op1.op = "<"$
- $op1 \rightarrow >$   
 $op1.op = ">"$
- $op1 \rightarrow <=$   
 $op1.op = "<="$
- $op1 \rightarrow >=$   
 $op1.op = ">="$
- $op1 \rightarrow ==$   
 $op1.op = "=="$
- $op1 \rightarrow !=$   
 $op1.op = "!="$
- $op4 \rightarrow *$   
 $op4.op = "*"$
- $op4 \rightarrow /$   
 $op4.op = "/"$
- $op4 \rightarrow \%$   
 $op4.op = "\%"$
- $op5 \rightarrow -$   
 $op5.op = "-"$
- $op5 \rightarrow \text{not}$   
 $op5.op = "\text{not}"$



- fun mkop1(op, opnd1, opnd2):
  - op = “<” → return menor(opnd1, opnd2)
  - op = “>” → return mayor(opnd1, opnd2)
  - op = “<=” → return menor\_igual(opnd1, opnd2)
  - op = “>=” → return mayor\_igual(opnd1, opnd2)
  - op = “==” → return igual(opnd1, opnd2)
  - op = “!=” → return no\_igual(opnd1, opnd2)
- fun mkop4(op, opnd1, opnd2):
  - op = “\*” → return mul(opnd1, opnd2)
  - op = “/” → return div(opnd1, opnd2)
  - op = “%” → return mod(opnd1, opnd2)
- fun mkop5(op, opnd):
  - op = “-” → return neg(opnd)
  - op = “not” → return not(opnd)

## 5 Especificación de proceso para imprimir

- imprime(prog(Bloq)):
  - imprime(Bloq)
- imprime(bloq(Decs, Insts)):
  - print “{”
  - nl
  - imprime(Decs)
  - imprime(Insts)
  - nl
  - print “}”
- imprime(si\_decs(DecsAux)):
  - imprime(DecsAux)
  - print “&&”

- `imprime(no_decs()): noop`
- `imprime(muchas_decs(DecsAux, Dec)):`  
`imprime(DecsAux)`  
`print “,”`  
`nl`  
`imprime(Dec)`
- `imprime(una_dec(Dec)):`  
`imprime(Dec)`
- `imprime(dec_var(T, Iden)):`  
`imprime(T)`  
`imprime(Iden)`
- `imprime(dec_tipo(T, Iden)):`  
`print “type”`  
`imprime(T)`  
`imprime(Iden)`
- `imprime(dec_proc(Iden, ParamsF, Bloq)):`  
`print “proc”`  
`imprime(Iden)`  
`print “(”`  
`imprime(ParamsF)`  
`print “)”`  
`imprime(Bloq)`
- `imprime(si_paramF(ParamsFL)):`  
`imprime(ParamsFL)`
- `imprime(no_paramF()): noop`
- `imprime(muchos_paramsF(ParamsFL, Param)):`  
`imprime(ParamsFL)`  
`print “,”`  
`imprime(Param)`

- `imprime(un_paramF(Param)):`  
`imprime(Param)`
- `imprime(param_ref(T, Iden)):`  
`imprime(T)`  
`print "&"`  
`imprime(Iden)`
- `imprime(param(T, Iden)):`  
`imprime(T)`  
`imprime(Iden)`
- `imprime(tipo_array(T, LitEnt)):`  
`imprime(T)`  
`print "["`  
`imprime(LitEnt)`  
`print "]"`
- `imprime(tipo_punt(T)):`  
`print "^"`  
`imprime(T)`
- `imprime(tipo_struct(LCampos)):`  
`print "struct"`  
`print "{"`  
`imprime(LCampos)`  
`print "}"`
- `imprime(muchos_campos(LCampos, Campo)):`  
`imprime(LCampos)`  
`print ","`  
`imprime(Campo)`
- `imprime(un_campo(Campo)):`  
`imprime(Campo)`

- `imprime(campo(T, Iden)):`  
`imprime(T)`  
`imprime(Iden)`
- `imprime(tipo_int()):`  
`print "int"`
- `imprime(tipo_real()):`  
`print "real"`
- `imprime(tipo_bool()):`  
`print "bool"`
- `imprime(tipo_string()):`  
`print "string"`
- `imprime(tipo_iden(Iden)):`  
`imprime(Iden)`
- `imprime(si_instr(InstAux)):`  
`imprime(InstAux)`
- `imprime(no_instr()):` `noop`
- `imprime(muchas_instr(InstsAux, Inst)):`  
`imprime(InstAux)`  
`print ";`  
`imprime(Inst)`
- `imprime(una_instr(Inst)):`  
`imprime(Inst)`
- `imprime(instr_expr(Exp)):`  
`print "@"`  
`imprime(Exp)`
- `imprime(instr_if(Exp, Bloq)):`  
`print "if"`  
`imprime(Exp)`  
`imprime(Bloq)`

- `imprime(instr_if_else(Exp, Bloq, Bloq))`:
  - `print "if"`
  - `imprime(Exp)`
  - `imprime(Bloq)`
  - `print "else"`
  - `imprime(Bloq)`
- `imprime(instr_while(Exp, Bloq))`:
  - `print "while"`
  - `imprime(Exp)`
  - `imprime(Bloq)`
- `imprime(instr_read(Exp))`:
  - `print "read"`
  - `imprime(Exp)`
- `imprime(instr_write(Exp))`:
  - `print "write"`
  - `imprime(Exp)`
- `imprime(instr_nl())`:
  - `print "nl"`
- `imprime(instr_new(Exp))`:
  - `print "new"`
  - `imprime(Exp)`
- `imprime(instr_del(Exp))`:
  - `print "delete"`
  - `imprime(Exp)`
- `imprime(instr_call(Iden, ParamsR))`:
  - `print "call"`
  - `imprime(Iden)`
  - `print "("`
  - `imprime(ParamsR)`
  - `print ")"`

- `imprime(instr_bloque(Bloq)):`  
`imprime(Bloq)`
- `imprime(si_paramsR(ParamsRL)):`  
`imprime(ParamsRL)`
- `imprime(no_paramsR()):` noop
- `imprime(muchos_paramsR(ParamsRL, Exp)):`  
`imprime(ParamsRL)`  
`print “,”`  
`imprime(Exp)`
- `imprime(un_paramsR(Exp)):`  
`imprime(Exp)`
- `imprime(asig(Exp1, Exp2)):`  
`imprimeExpBin(Exp1, “=”, Exp2, 1, 0)`
- `imprime(menor(Exp1, Exp2)):`  
`imprimeExpBin(Exp1, “<”, Exp2, 1, 2)`
- `imprime(mayor(Exp1, Exp2)):`  
`imprimeExpBin(Exp1, “>”, Exp2, 1, 2)`
- `imprime(menor_igual(Exp1, Exp2)):`  
`imprimeExpBin(Exp1, “<=”, Exp2, 1, 2)`
- `imprime(mayor_igual(Exp1, Exp2)):`  
`imprimeExpBin(Exp1, “>=”, Exp2, 1, 2)`
- `imprime(igual(Exp1, Exp2)):`  
`imprimeExpBin(Exp1, “==”, Exp2, 1, 2)`
- `imprime(no_igual(Exp1, Exp2)):`  
`imprimeExpBin(Exp1, “!=”, Exp2, 1, 2)`
- `imprime(suma(Exp1, Exp2)):`  
`imprimeExpBin(Exp1, “+”, Exp2, 2, 3)`

- `imprime(resta(Exp1, Exp2)):`  
`imprimeExpBin(Exp1, "-", Exp2, 3, 3)`
- `imprime(and(Exp1, Exp2)):`  
`imprimeExpBin(Exp1, "and", Exp2, 4, 3)`
- `imprime(or(Exp1, Exp2)):`  
`imprimeExpBin(Exp1, "or", Exp2, 4, 4)`
- `imprime(mult(Exp1, Exp2)):`  
`imprimeExpBin(Exp1, "*", Exp2, 4, 5)`
- `imprime(div(Exp1, Exp2)):`  
`imprimeExpBin(Exp1, "/", Exp2, 4, 5)`
- `imprime(mod(Exp1, Exp2)):`  
`imprimeExpBin(Exp1, "%", Exp2, 4, 5)`
- `imprime(negativo(Exp)):`  
`imprimeExpUnarioPrefijo("-", 5)`
- `imprime(not(Exp)):`  
`imprimeExpUnarioPrefijo("not", 5)`
- `imprime(index(Exp1, Exp2)):`  
`imprimeOpnd(Exp1, 6)`  
`print "[`  
`imprime(Exp)`  
`print "]"`
- `imprime(acceso(Exp, Iden)):`  
`imprimeOpnd(Exp1, 6)`  
`print "."`  
`imprime(Iden)`
- `imprime(indireccion(Exp)):`  
`imprimeOpnd(Exp1, 6)`  
`print "^"`

- `imprime(lit_ent(N)):`  
    `print N`
- `imprime(lit_real(R)):`  
    `print R`
- `imprime(true()):`  
    `print "true"`
- `imprime(false()):`  
    `print "false"`
- `imprime(lit_cadena(Cad)):`  
    `print Cad`
- `imprime(iden(Id)):`  
    `print Id`
- `imprime(null()):`  
    `print "null"`
- `imprimeOpnd(Opnd,MinPrior):`  
    `if prioridad(Opnd) < MinPrior`  
        `print "("`  
    `end if`  
    `imprime(Opnd)`  
    `if prioridad(Opnd) < MinPrior`  
        `print ")"`  
    `end if`
- `imprimeExpBin(Opnd0,Op,Opnd1,np0,np1):`  
    `imprimeOpnd(Opnd0,np0)`  
    `print " " ++ Op ++ " "`  
    `imprimeOpnd(Opnd1,np1)`
- `imprimeExpUnarioPrefijo(Opnd, Op, np):`  
    `print Op ++ " "`  
    `imprimeOpnd(Opnd,np)`



- prioridad(asig(.,.)): return 0
- prioridad(menor(.,.)): return 1
- prioridad(mayor(.,.)): return 1
- prioridad(menor\_igual(.,.)): return 1
- prioridad(mayor\_igual(.,.)): return 1
- prioridad(igual(.,.)): return 1
- prioridad(no\_igual(.,.)): return 1
- prioridad(suma(.,.)): return 2
- prioridad(resta(.,.)): return 2
- prioridad(and(.,.)): return 3
- prioridad(or(.,.)): return 3
- prioridad(mul(.,.)): return 4
- prioridad(div(.,.)): return 4
- prioridad(mod(.,.)): return 4
- prioridad(negativo(.)): return 5
- prioridad(not(.)): return 5
- prioridad(index(.,.)): return 6
- prioridad(acceso(.,.)): return 6
- prioridad(indireccion(.)): return 6