Fase 3: Desarrollo de constructores de ASTs para Tiny

G01
Esther Babon Arcauz
Pablo Campo Gómez
Claudia López-Mingo Moreno
José Antonio Ruiz Heredia

- [x] ~~Especificación de la sintaxis abstracta de Tiny mediante la enumeración de las signaturas (cabeceras) de las funciones constructoras de ASTs.~~
- [x] ~~Especificación del constructor de ASTs mediante una gramática s-atribuida.~~
- [x] ~~Acondicionamiento de dicha especificación para permitir la implementación descendente.~~
- [ ] Especificación de un procesamiento que imprima 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 .

- [x] ~~2. Corrección de esta entrega Pablo (y esther y clau)~~

- [ ] 3. Acabar especificación del procesamiento punto 4 . Pablo

- [ ] 4. Una implementación orientada a objetos en Java de la sintaxis abstracta de Tiny, preparada tanto para soportar programación recursiva, como para soportar procesamiento mediante el patrón visitante. Pablo (Depende de puntos 2,3)

- [ ] 5. Una implementación descendente del constructor de ASTs desarrollada con javacc. Esther (Depende de puntos 2,3)

- [ ] 6. Una implementación ascendente de dicho constructor desarrollada con CUP y jflex. Esther (Depende de puntos 2,3)

- [ ] 7. Tres implementaciones del procesamiento pedido: (i) una utilizando programación recursiva; (ii) una utilizando el patrón intérprete; (iii) una tercera utilizando el patrón visitante (Depende de puntos 4, 5, 6). Clau

- [ ] 8. Un programa principal que integre ambos constructores, y también los procesamientos desarrollados. Dicho programa recibirá como argumentos (i) el archivo a analizar; (ii) una opción opc que indique el constructor de ASTs a aplicar (si op es desc el constructor a aplicar será el descendente; si es asc será el

ascendente); y (iii) una opción opp que indica que estilo de procesamiento a aplicar (si op es rec se aplicará programación recursiva, si op es int se aplicará el procesamiento basado en el patrón intérprete, si es vis el basado en el patrón visitante). El programa producirá como salida, bien un mensaje legible del primer error (léxico o sintáctico) detectado, bien una impresión, en los términos indicados, del programa leído en caso de que no se hayan detectado errores. (Depende de puntos 4,5,6,7) Jose

Índice:

1. Especificación de la sintaxis abstracta de Tiny mediante la enumeración de las signaturas (cabeceras) de las funciones constructoras de ASTs.

No terminal	Género
programa	Prog
bloque	Bloq
declaraciones_opt	DecsOpt
declaraciones	LDecs
istrucciones_opt	InsOpt
instrucciones	LIns
instruccion	Ins

declaracion	Dec
tipo	T
parametros_formales_opt	PFormOpt
parametros_formales	LPForm
parametro_formal	PForm
campos	LCamp
campo	Camp
parametros_reales_opt	PRealOpt
parametros_reales	LPReal
E0,E1,E2,E3,E4,E5,E6	Exp

GIC	Constructora de sintaxis abstracta
programa → bloque EOF	prog: Bloq x EOF → Prog
bloque → { declaraciones_opt instrucciones_opt }	bloq: DecsOpt x InsOpt → Bloq
declaraciones_opt → declaraciones && declaraciones_opt → ε	si_decs: LDecs → DecsOpt no_decs: → DecsOpt
istrucciones_opt → instrucciones instrucciones_opt → ε	si_ins: LIns → InsOpt no_ins: → InsOpt
declaraciones → declaraciones ; declaracion declaraciones → declaracion	muchas_decs: LDecs x Dec → LDecs una_dec: Dec → LDecs
declaracion → tipo identificador declaracion → type tipo identificador declaracion → proc identificador (parametros_formales_opt) bloque	dec_var: T x string → Dec dec_tipo: T x string → Dec dec_proc: string x PFormOpt x Bloq → Dec
parametros_formales_opt → parametros_formales parametros_formales_opt → ε	si_pform: LPForm → PFormOpt no_pform: → PFormOpt

parametros_formales → parametros_formales , parametro_formal parametros_formales → parametro_formal	muchas_pforms: LPForm x PForm → LPForm una_pform: PForm → LPForm
parametro_formal → tipo and_opt identificador and_opt → & and_opt → ε	pform_ref: T x string → PForm pform_no_ref: T x string → PForm
tipo → tipo [literalEntero] tipo → tipo1 tipo1 → ^ tipo1 tipo1 → tipo2 tipo2 → identificador tipo2 → struct { campos } tipo2 → int tipo2 → real tipo2 → bool tipo2 → string	array: T x string → T puntero: T → T iden: string → T struct: LCamp → T lit_ent: → T lit_real: → T lit_bool: → T lit_string: → T
campos → campos , campo campos → campo	muchos_camp: LCamp x Camp → LCamp un_camp: Camp → LCamp
campo → tipo identificador	camp: T x string → Camp
instrucciones → instrucciones ; instruccion instrucciones → instruccion	muchas_ins: LIns x Ins → LIns una_ins: Ins → LIns
instruccion → @ E0 instruccion → if E0 bloque instruccion → if E0 bloque else bloque instruccion → while E0 bloque instruccion → read E0 instruccion → write E0 instruccion → nl instruccion → new E0 instruccion → delete E0 instruccion → call identificador (parametros_reales_opt) instruccion → bloque	ins_asig: Exp → Ins ins_if: Exp x Bloq → Ins ins_if_else: Exp x Bloq x Bloq → Ins ins_while: Exp x Bloq → Ins ins_read: Exp → Ins ins_write: Exp → Ins ins_nl: → Ins ins_new: Exp → Ins ins_delete: Exp → Ins ins_call: string x PRealOpt → Ins ins_bloque: Bloq→ Ins
parametros_reales_opt -> parametros_reales parametros_reales_opt → ε	si_preal: LPReal → PRealOpt no_preal: → PRealOpt
parametros_reales -> parametros_reales , E0 parametros_reales -> E0	muchos_preal: LPReal x Exp → LPReal un_preal: Exp → LPReal

E0 → E1 = E0 E0 → E1 E1 → E1 op1 E2 E1 → E2 E2 → E2 + E3 E2 → E3 - E3 E2 → E3 E3 → E4 op3 E3 → E4 E4 → E4 op4 E5 E4 → E5 E5 → op5 E5 E5 → E6 E6 → E6 op6 E6 → E7 E7 → literalEntero E7 → literalReal E7 → literalBool E7 → literalCadena E7 → identificador E7 → null E7 → (E0) op1 → > op1 → >= op1 → < op1 → <= op1 → == op1 → != op3 → and E3 op3 → or E4 op4 → * op4 → / op4 → % op5 → - op5 → not op6 → [E0] op6 → .identificador op6 → ^	exp_bin: Expbin → Exp exp_un: Expun → Exp asig: Exp x Exp → Expbin mayor: Exp x Exp → Expbin menor: Exp x Exp → Expbin mayorIgual: Exp x Exp → Expbin menorIgual: Exp x Exp → Expbin igual: Exp x Exp → Expbin desigual: Exp x Exp → Expbin suma: Exp x Exp → Expbin resta: Exp x Exp → Expbin and: Exp x Exp → Expbin or: Exp x Exp → Expbin mul: Exp x Exp → Expbin div: Exp x Exp → Expbin mod: Exp x Exp → Expbin neg: Exp→ Expun not: Exp → Expun acceso_array: Exp x Exp → Exp acceso_campo: Exp x string → Exp acceso_puntero: Exp → Exp exp_litEntero: → Exp exp_litReal: → Exp exp_litBoolTrue: → Exp exp_litBoolFalse: → Exp exp_litCadena: string→ Exp exp_identificador: string→ Exp exp_null: → Exp

2. Especificación del constructor de ASTs mediante una gramática s-atribuida.

programa → bloque EOF

programa.a = prog(bloque.a)

bloque → { declaraciones_opt instrucciones_opt }
bloque.a = bloq(declaraciones_opt.a, instrucciones_opt.a)

declaraciones_opt → declaraciones &&
declaraciones_opt.a = si_decs(declaraciones.a)

declaraciones_opt → ε
declaraciones_opt.a = no_decs()

istrucciones_opt → instrucciones
instrucciones_opt.a = si_ins(instrucciones.a)

instrucciones_opt → ε
instrucciones_opt.a = no_ins()

declaraciones → declaraciones ; declaracion
declaraciones0.a = muchas_decs(declaraciones1.a, declaracion.a)

declaraciones → declaracion
declaraciones.a = una_dec(declaracion.a)

declaracion → tipo identificador
declaracion.a = dec_var(tipo.a, identificador.lex)

declaracion → type tipo identificador
declaracion.a = dec_tipo(tipo.a, identificador.lex)

declaracion → proc identificador (parametros_formales_opt) bloque
declaracion.a = dec_proc(identificador.lex, parametros_formales_opt.a, bloque.a)

parametros_formales_opt → parametros_formales
parametros_formales_opt.a = si_pform(parametros_formales.a)

parametros_formales_opt → ε
parametros_formales_opt.a = no_pform()

parametros_formales → parametros_formales , parametro_formal
parametros_formales0.a = muchas_pforms(parametros_formales1.a, parametro_formal.a)

parametros_formales → parametro_formal
parametros_formales.a = una_pform(parametro_formal.a)

parametro_formal → tipo and_opt identificador
parametro_formal.a = pform(tipo.a, and_opt.a, identificador.lex)
and_opt → &
and_opt.a = pform_ref()
and_opt → ε
and_opt.a = pform_no_ref()

tipo → tipo [literalEntero]
tipo0.a = array(tipo1.a, literalEntero.lex)

tipo → tipo1
tipo.a = tipo1.a

tipo1 → ^ tipo1
tipo10.a = puntero(tipo11.a)

tipo1 → tipo2
tipo1.a = tipo2.a

tipo2 → identificador
tipo2.a = iden(identificador.lex)

tipo2 → struct { campos }
tipo2.a = struct(campos.a)

tipo2 → int
tipo2.a = lit_ent()

tipo2 → real
tipo2.a = lit_real()

tipo2 → bool
tipo2.a = lit_bool()

tipo2 → string
tipo2.a = lit_string()

campos → campos , campo
campos0.a = muchos_camp(campos1.a, campo.a)

campos → campo
campos.a = un_camp(campo.a)

campo → tipo identificador
campo.a = camp(tipo.a, identificador.lex)

instrucciones → instrucciones ; instruccion
instrucciones0.a = muchas_ins(instrucciones1.a, instruccion.a)

instrucciones → instruccion
instrucciones.a = una_ins(instruccion.a)

instruccion → @ E0
instruccion.a = ins_asig(E0.a)

instruccion → if E0 bloque
instruccion.a = ins_if(E0.a, bloque.a)

instruccion → if E0 bloque else bloque
instruccion.a = ins_if_else(E0.a, bloque0.a, bloque1.a)

instruccion → while E0 bloque
instruccion.a = ins_while(E0.a, bloque.a)

instruccion → read E0
instruccion.a = ins_read(E0.a)

instruccion → write E0
instruccion.a = ins_write(E0.a)

instruccion → nl
instruccion.a = ins_nl()

instruccion → new E0
instruccion.a = ins_new(E0.a)

instruccion → delete E0
instruccion.a = ins_delete(E0.a)

instruccion → call identificador (parametros_reales_opt)
instruccion.a = ins_call(identificador.lex, parametros_reales_opt.a)

instruccion → bloque

instruccion.a = ins_bloque(bloque.a)

parametros_reales_opt → parametros_reales
parametros_reales_opt.a = si_preal(parametros_reales.a)

parametros_reales_opt → ε
parametros_reales_opt.a = no_preal()

parametros_reales → parametros_reales , E0
parametros_reales0.a = muchos_preal(parametros_reales1.a, E0.a)

parametros_reales → E0
parametros_reales.a = un_preal(E0.a)

E0 → E1 = E0
E00.a = asig(E1.a, E01.a)

E0 → E1
E0.a = E1.a

E1 → E1 op1 E2
E10.a = mkop2(op1.op, E11.a, E2.a)

E1 → E2
E1.a = E2.a

E2 → E2 + E3
E20.a = mkop2(“+”, E21.a, E3.a)

E2 → E3 - E3
E2.a =mkop2(“-”, E30.a, E31.a)

E2 → E3
E2.a = E3.a

E3 → E4 and E3
E30.a = and(E4.a, E31.a)

E3 → E4 or E4
E3.a = or(E40.a, E41.a)

E3 → E4
E3.a = E4.a

E4 → E4 op4 E5
E40.a = mkop2(op4.op, E41, E5)

E4 → E5
E4.a = E5.a

E5 → op5 E5
E50.a = mkop1(op5.op, E51.a)

E5 → E6
E5.a = E6.a

E6 → E6 [E0]
E60.a = acceso_array(E61.a, E0.a)

E6 → E6 . identificador
E60.a = acceso_campo(E61.a, identificador.lex)

E6 → E6 ^
E60.a = acceso_puntero(E61.a)

E6 → E7
E6.a = E7.a

E7 → literalEntero
E7.a = exp_litEntero(literalEntero.lex)

E7 → literalReal
E7.a = exp_litReal(literalEntero.lex)

E7→ literalTrue
E7.a = exp_litBoolTrue()

E7 → literalFalse
E7.a = exp_litBoolFalse()

E7 → literalCadena
E7.a = exp_litCadena(literalCadena.lex)

E7 → identificador
E7.a = exp_litIdentificador(identificador.lex)

E7 → null

E7.a = exp_null()

E7 → (E0)
E7.a = E0.a

op1 → >
op1.op = “>”

op1 → >=
op1.op = “>=”

op1 → <
op1.op = “<”

op1 → <=
op1.op = “<=”

op1 → ==
op1.op = “==”

op1 → !=
op1.op = “!=”

**op4 → *
**op4.op = “*”

op4 → /
op4.op = “/”

op4 → %
op4.op = “%”

op5 → -
op5.op = “-”

op5 → not
op5.op = “not”

```
fun mkop2(op, opnd1,opnd2):
        op = "+" → return suma(opnd1, opnd2)
        op = "-" → return resta(opnd1, opnd2)
        op = "*" → return mul(opnd1, opnd2)
        op = "/" → return div(opnd1, opnd2)
        op = “%” →  return mod(opnd1, opnd2)
        op = “=” →  return asig(opnd1, opnd2)
        op = “>” →  return mayor(opnd1, opnd2)
        op = “<” →  return menor(opnd1, opnd2)
        op = “>=” →  return mayorIgual(opnd1, opnd2)
        op = “<=” →  return menorIgual(opnd1, opnd2)
        op = “==” →  return igual(opnd1, opnd2)
        op = “!=” →  return desigual(opnd1, opnd2)


fun mkop1(op, opnd1):
        op = “-” →  return neg(opnd1)
        op = “not” →  return not(opnd1)
```

3. Acondicionamiento de dicha especificación para permitir la implementación descendente.

Gramática s-atribuida	Acondicionado
programa → bloque EOF programa.a = prog(bloque.a)	
bloque → { declaraciones_opt instrucciones_opt } bloque.a = bloq(declaraciones_opt.a, instrucciones_opt.a)	
declaraciones_opt → declaraciones && declaraciones_opt.a = si_decs(declaraciones.a) **declaraciones_opt → ε** declaraciones_opt.a = no_decs()	
istrucciones_opt → instrucciones instrucciones_opt.a = si_ins(instrucciones.a) **instrucciones_opt → ε** instrucciones_opt.a = no_ins()	
declaraciones → declaraciones ; declaracion declaraciones0.a = muchas_decs(declaraciones1.a, declaracion.a) **declaraciones → declaracion** declaraciones.a = una_dec(declaracion.a)	**declaraciones → declaracion declaraciones'** declaraciones'.h = una_dec(declaracion.a) declaraciones.a = declaraciones'.a **declaraciones' → ; declaracion declaraciones'** declaraciones'1.h = muchas_decs(declaraciones'0.h, declaracion.a) declaraciones'0.a = declaraciones'1.a **declaraciones' → ε** declaraciones'.a = declaraciones'.h
declaracion → tipo identificador declaracion.a = dec_var(tipo.a, identificador.lex) **declaracion → type tipo identificador** declaracion.a = dec_tipo(tipo.a, identificador.lex) **declaracion→proc identificador (parametros_formales_opt) bloque** declaracion.a = dec_proc(identificador.lex, parametros_formales_opt.a, bloque.a)	
parametros_formales_opt → parametros_formales parametros_formales_opt.a = si_pform(parametros_formales.a) **parametros_formales_opt → ε** parametros_formales_opt.a = no_pform()	
parametros_formales → parametros_formales , parametro_formal	**parametros_formales → parametro_formal parametros_formales'**

parametros_formales0.a = muchas_pforms(parametros_formales1.a, parametro_formal.a) **parametros_formales → parametro_formal** parametros_formales.a = una_pform(parametro_formal.a)	parametros_formales'.h = una_pform(parametro_formal.a) parametros_formales.a = parametros_formales'.a **parametros_formales' → , parametro_formal parametros_formales'** parametros_formales'1.h = muchas_pforms(parametros_formales'0.h, parametro_formal.a) parametros_formales'0.a = parametros_formales'1.a **parametros_formales' → ε** parametros_formales'.a = parametros_formales'.h
parametro_formal → tipo and_opt identificador parametro_formal.a = pform(tipo.a, and_opt.a, identificador.lex) **and_opt → &** and_opt.a = pform_ref() **and_opt → ε** and_opt.a = pform_no_ref()	
tipo → tipo [literalEntero] tipo0.a = array(tipo1.a, literalEntero.lex) **tipo → tipo1** tipo.a = tipo1.a	**tipo → tipo1 tipo'** tipo'.h = tipo1.a tipo.a = tipo'.a **tipo' → [literalEntero] tipo'** tipo'1.h = array(tipo'0.h, literalEntero.lex) tipo'0.a = tipo'1.a **tipo' → ε** tipo'.a = tipo'.h
tipo1 → ^ tipo1 tipo10.a = puntero(tipo11.a) **tipo1 → tipo2** tipo1.a = tipo2.a	
tipo2 → identificador tipo2.a = iden(identificador.lex) **tipo2 → struct { campos }** tipo2.a = struct(campos.a) **tipo2 → int** tipo2.a = lit_ent() **tipo2 → real** tipo2.a = lit_real() **tipo2 → bool** tipo2.a = lit_bool() **tipo2 → string** tipo2.a = lit_string()	
campos → campos , campo	**campos → campo campos'**

campos0.a = muchos_camp(campos1.a, campo.a) **campos → campo** campos.a = un_camp(campo.a)	campos'.h = un_camp(campo.a) campos.a = campos'.a **campos' → , campo campos'** campos'1.h = muchos_camp(campos'0.h, campo.a) campos'0.a = campos'1.a **campos' → ε** campos'.a = campos'.h
campo → tipo identificador campo.a = camp(tipo.a, identificador.lex)	
instrucciones → instrucciones ; instruccion instrucciones0.a = muchas_ins(instrucciones1.a, instruccion.a) **instrucciones → instruccion** instrucciones.a = una_ins(instruccion.a)	**instrucciones → instruccion instrucciones'** instrucciones'.h = una_ins(instruccion.a) instrucciones.a = instrucciones'.a **instrucciones' → ; instrucción instrucciones'** instrucciones'1.h = muchas_ins(instrucciones'0.h, instrucción.a) instrucciones'0.a = instrucciones'1.a **instrucciones' → ε** instrucciones'.a = instrucciones'.h
instruccion → @ E0 instruccion.a = ins_asig(E0.a) **instruccion → if E0 bloque** instruccion.a = ins_if(E0.a, bloque.a) **instruccion → if E0 bloque else bloque** instruccion.a = ins_if_else(E0.a, bloque0.a, bloque1.a) **instruccion → while E0 bloque** instruccion.a = ins_while(E0.a, bloque.a) **instruccion → read E0** instruccion.a = ins_read(E0.a) **instruccion → write E0** instruccion.a = ins_write(E0.a) **instruccion → nl** instruccion.a = ins_nl() **instruccion → new E0** instruccion.a = ins_new(E0.a) **instruccion → delete E0** instruccion.a = ins_delete(E0.a) **instruccion → call identificador (parametros_reales_opt)** instruccion.a = ins_call(identificador.lex, parametros_reales_opt.a) **instruccion → bloque** instruccion.a = ins_bloque(bloque.a)	**instruccion → @ E0** instruccion.a = ins_asig(E0.a) **instruccion → parteif restoif** restoif.h = parteif.a instruccion.a = restoif.a **parteif → if E0 bloque** parteif.a = ins_if(E0.a, bloque.a) **restoif → ε** restoif.a = restoif.h **restoif → else bloque** restoif.a = ins_if_else(restoif.h, bloque.a) **instruccion → while E0 bloque** instruccion.a = ins_while(E0.a, bloque.a) **instruccion → read E0** instruccion.a = ins_read(E0.a) **instruccion → write E0** instruccion.a = ins_write(E0.a) **instruccion → nl** instruccion.a = ins_nl() **instruccion → new E0** instruccion.a = ins_new(E0.a) **instruccion → delete E0** instruccion.a = ins_delete(E0.a) **instruccion → call identificador (parametros_reales_opt)** instruccion.a = ins_call(identificador.lex, parametros_reales_opt.a)

	instruccion → bloque instruccion.a = ins_bloque(bloque.a)
parametros_reales_opt → parametros_reales parametros_reales_opt.a = si_preal(parametros_reales.a) **parametros_reales_opt → ε** parametros_reales_opt.a = no_preal()	
parametros_reales → parametros_reales , E0 parametros_reales0.a = muchos_preal(parametros_reales1.a, E0.a) **parametros_reales → E0** parametros_reales.a = un_preal(E0.a)	**parametros_reales → E0 parametros_reales'** parametros_reales'.h = un_preal(E0.a) parametros_reales.a = parametros_reales'.a **parametros_reales' → , E0 parametros_reales'** parametros_reales'1.h = muchos_preal(parametros_reales'0.h, E0.a) parametros_reales'0.a = parametros_reales'1.a **parametros_reales' → ε** parametros_reales'.a = parametros_reales'.h
E0 → E1 = E0 E00.a = asig(E1.a, E01.a) **E0 → E1** E0.a = E1.a	**E0 → E1 asignacion** asignacion.h = E1.a E0.a = asignacion.a **asignacion → = E0** asignacion.a = asig(asignacion.h, E0.a) **asignacion → ε** asignacion.a = asignacion.h
E1 → E1 op1 E2 E10.a = mkop2(op1.op, E11.a, E2.a) **E1 → E2** E1.a = E2.a	**E1 → E2 E1'** E1'.h = E2.a E1.a = E1'.a **E1' → op1 E2 E1'** E1'1.h = mkop2(E1'0.h , op1.op, E2.a) E1'0.a = E1'1.a **E1' → ε** E1'.a = E1'.h
E2 → E2 + E3 E20.a = mkop2("+", E21.a, E3.a) **E2 → E3 - E3** E2.a =mkop2("-", E30.a, E31.a) **E2 → E3** E2.a = E3.a	**E2 → E3 minus E2'** minus.h = E3.a E2'.H = minus.a E2.a = E2'.a **E2' → + E3 E2'** E2'1.h = mkop2(E2'1.h, "+", E3.a E2'0.a = E2'1.a **E2' → ε** E2'.a = E2'.h **minus → - E3** minus.a = mkop2(minus.h, "-", E3.a) **minus → ε** minus.a = minus.h

E3 → E4 and E3 E30.a = and(E4.a, E31.a) **E3 → E4 or E4** E3.a = or(E40.a, E41.a) **E3 → E4** E3.a = E4.a	**E3 → E4 and_or** and_or.h = E4.a E3.a = and_or.a **and_or → and E3** and_or.a = and(and_or.h, E3.a) **and_or → or E4** and_or.a = or(and_or.h E4.a) **and_or → ε** and_or.a = and_or.h
E4 → E4 op4 E5 E40.a = mkop2(op4.op, E41.a, E5.a) **E4 → E5** E4.a = E5.a	**E4 → E5 E4'** E4'.h = E5.a E4.a → E4'.a **E4' → op4 E5 E4'** E4'1.h = mkop2(E4'0.h, op4.op, E5.a) E4'0.a = E4'1.a **E4' → ε** E4'.a = E4'.h
E5 → op5 E5 E50.a = mkop1(op5.op, E51.a) **E5 → E6** E5.a = E6.a	
E6 → E6 [E0] E60.a = acceso_array(E61.a, E0.a) **E6 → E6 . identificador** E60.a = acceso_campo(E61.a, identificador.lex) **E6 → E6 ^** E60.a = acceso_puntero(E61.a) **E6 → E7** E6.a = E7.a	**E6 → E7 E6'** E6'.h = E7.a E6.a = E6'.a **E6' → resto6 E6'** E6'1.h = resto6.a E6'0.a = E6'1.a **E6' → ε** E6'.a = E6'.h **resto6 → [E0]** resto6.a = acceso_array(resto6.h, E0.a) **resto6 → . identificador** resto6.a = acceso_campo(resto6.h, identificador.lex) **resto6 → ^** resto6.a = acceso_puntero(resto6.h)
E7 → literalEntero E7.a = exp_litEntero(literalEntero.lex) **E7 → literalReal** E7.a = exp_litReal(literalReal.lex) **E7→ literalTrue** E7.a = exp_litBoolTrue() **E7 → literalFalse** E7.a = exp_litBoolFalse() **E7 → literalCadena**	

E7.a = exp_litCadena(literalCadena.lex) **E7 → identificador** E7.a = exp_litIdentificador(identificador.lex) **E7 → null** E7.a = exp_null() **E7 → (E0)** E7.a = E0.a	
op1 → > op1.op = “>” **op1 → >=** op1.op = “>=” **op1 → <** op1.op = “<” **op1 → <=** op1.op = “<=” **op1 → ==** op1.op = “==” **op1 → !=** op1.op = “!=”	
op4 → * op4.op = “*” **op4 → /** op4.op = “/” **op4 → %** op4.op = “%”	
op5 → - op5.op = “-” **op5 → not** op5.op = “not”	

4. Especificación de un procesamiento que imprima los tokens del programa leído

Procesamiento del lenguaje Tiny orientado a realizar una impresión bonita del programa. En concreto que:

- Las expresiones deben imprimirse con el mínimo número necesario de paréntesis
- Un token leído por cada línea
- Las palabras reservadas se escriben en minúsculas y con < >
- Las palabras reservadas son: int real bool string and or not null true false proc if else while struct new delete read write nl type call
- El fin de fichero se escribirá como <EOF>

```
imprimeOpnd(Opnd,MinPrior):
        if prioridad(Opnd) < MinPrior
                print “(“
        imprime(Opnd)
        if prioridad(Opnd) < MinPrior
                print “)“


prioridad(asig(_,_)):  return 0
prioridad(mayor(_,_)): return 1
prioridad(menor(_,_)): return 1
prioridad(mayorIgual(_,_)): return 1
prioridad(menorIgual(_,_)): return 1
prioridad(igual(_,_)): return 1
prioridad(desigual(_,_)): 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(neg(_)): return 5
prioridad(not(_)): return 5
prioridad(acceso_array(_,_)): return 6
prioridad(acceso_campo(_,_)): return 6
prioridad(acceso_puntero(_,_)): return 6
prioridad(exp_litEntero(_)): return 7
prioridad(exp_litReal(_)): return 7
prioridad(exp_litBoolTrue(_)): return 7
prioridad(exp_litBoolFalse(_)): return 7
prioridad(exp_litCadena(_)): return 7
prioridad(exp_identificador(_)): return 7
prioridad(exp_null(_)): return 7


imprime(prog(Bloq)):
        imprime(Bloq)
        print “<EOF>”
```

```
imprime(bloq(DecsOpt, InsOpt)):
        print “{“
        imprime(DecsOpt)
        imprime(InsOpt)
        print “}”


imprime(si_decs(LDecs)):
        imprime(LDecs)
        print ”&&”


imprime(no_decs()):
        skip

imprime(muchas_decs(LDecs, Dec)):
        imprime(LDecs)
        print “;”
        imprime(Dec)


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


imprime(dec_var(T, string)):
        imprime(T)
        print string

imprime(dec_tipo(T, string)):
        print <type>
        imprime(T)
        print string


imprime(dec_proc(string, PFormOpt, Bloq)):
        print <proc>
        print string
        print “(“
        imprime(PFormOpt)
        print “)”
        imprime(Bloq)

imprime(si_pform(LPForm):
        imprime(LPForm)

imprime(no_pform()):
        skip

imprime(muchas_pforms(LPForm, PForm)):
        imprime(LPForm)
        print “,”
        imprime(PForm)
```

```
imprime(una_pform(PForm)):
        imprime(PForm)

imprime(pform_ref(T, string)):
        imprime(T)
        print “&”
        print string

imprime(pform_no_ref(T, string)):
        imprime(T)
        print string

imprime(array(T, string)):
        imprime(T)
        print “[” + string + “]”


imprime(puntero(T)):
        imprime(T)
        print ”^”

imprime(iden(string)):
        print string

imprime(struct(LCamp)):
        print <struct>
        print “ {“
        imprime(LCamp)
        print “}”


imprime(lit_ent(string)):
        print <int>

imprime(lit_real(string)):
        print <real>

imprime(lit_bool(string)):
        print<bool>

imprime(lit_string(string)):
        print<string>

imprime(muchos_camp(LCamp,  Camp)):
        imprime(LCamp)
        print “,”
        imprime(Camp)

imprime(un_camp(Camp)):
        imprime(Camp)

imprime(camp(T, string)):
        imprime(T)
```

```
        print string

imprime(Si_ins(LIns)):
        imprime(LIns)


imprime(no_ins()):
        skip

imprime(muchas_ins(LIns, Ins)):
        imprime(LIns)
        print ";"
        imprime(Ins)

imprime(una_ins(Ins)):
        imprime(Ins)

imprime(ins_asig(Exp)):
        print "@"
        imprime(Exp)

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

imprime(ins_if_else(Exp, Bloq, Bloq)):
        print "<if>"
        imprime(Exp)
        imprime(Bloq)
        print "<else>"
        imprime(Bloq)

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

imprime(ins_read(Exp)):
        print "<read>"
        imprime(Exp)

imprime(ins_write(Exp)):
        print "<write>"
        imprime(Exp)

imprime(ins_nl()):
        print "<nl>"
```

```
imprime(ins_new(Exp)):
        print “<new>”
        imprime(Exp)

imprime(ins_delete(Exp)):
        print “<delete>”
        imprime(Exp)

imprime(ins_call(string, PRealOpt)):
        print “<call>”
        print string
        print “(“
        imprime(PRealOpt)
        print “)”

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

imprime(si_preal(LPReal)):
        imprime(LPreal)

imprime(no_preal()):
        skip

imprime(muchos_preal(LPReal, Exp)):
        imprime(LPReal)
        print “,”
        imprime(Exp)

imprime(un_preal(Exp)):
        imprime(Exp)

imprimeExpBin(Exp0, Exp1, string, p0, p1)):
        imprimeOpnd(Exp0, p0)
        print string
        imprimeOpnd(Exp1,p1)

imprime(asig(Exp0, Exp1)):
        imprimeExpBin(Exp0, Exp1, “=”, 1, 0)

imprime(mayor(Exp0, Exp1)):
        imprimeExpBin(Exp0, Exp1, “>”, 1, 2)

imprime(menor(Exp0, Exp1)):
        imprimeExpBin(Exp0, Exp1, “<”, 1, 2)
```

```
imprime(mayorIgual(Exp0, Exp1)):
        imprimeExpBin(Exp0, Exp1, “>=”, 1, 2)

imprime(menorIgual(Exp0, Exp1)):
        imprimeExpBin(Exp0, Exp1, “<=”, 1, 2)

imprime(igual(Exp0, Exp1)):
        imprimeExpBin(Exp0, Exp1, “==”, 1, 2)

imprime(desigual(Exp0, Exp1)):
        imprimeExpBin(Exp0, Exp1, “!=”, 1, 2)

imprime(suma(Exp0, Exp1)):
        imprimeExpBin(Exp0, Exp1, “+”, 2, 3)

imprime(resta(Exp0, Exp1)):
        imprimeExpBin(Exp0, Exp1, “-”, 3, 3)

imprime(and(Exp0, Exp1)):
        imprimeExpBin(Exp0, Exp1, “<and>”, 4, 3)

imprime(or(Exp0, Exp1)):
        imprimeExpBin(Exp0, Exp1, “<or>”, 4, 4)

imprime(mul(Exp0, Exp1)):
        imprimeExpBin(Exp0, Exp1, “*”, 4, 5)

imprime(div(Exp0, Exp1)):
        imprimeExpBin(Exp0, Exp1, “/”, 4, 5)

imprime(mod(Exp0, Exp1)):
        imprimeExpBin(Exp0, Exp1, “%”, 4, 5)

imprimeExpUn(Exp, string, p)):
        print string
        imprimeOpnd(Exp, p)

imprime(neg(Exp)):
        imprimeExpUn(Exp, “-”, 5)

imprime(not(Exp)):
        imprimeExpUn(Exp, “<not>”, 5)

imprime(acceso_array(Exp0, Exp1)):
        imprimeOpnd(Exp0, 6)
        print “ [ ”
        imprime(Exp1)
        print “ ] ”
```

```
imprime(acceso_campo(Exp, id)):
        imprimeOpnd(Exp, 6)
        print ". "
        imprime(iden(id))

imprime(acceso_puntero(Exp)):
        imprimeOpnd(Exp, 6)
        print "^ "

imprime(exp_litEntero(N)):
        print N

imprime(exp_litReal(R)):
        print R

imprime(exp_litCadena(Id)):
        print Id

imprime(exp_Identificador(Id)):
        print Id

imprime(exp_litBoolTrue()):
        print "<true>"

imprime(exp_litBoolFalse()):
        print "<false>"

imprime(exp_null()):
        print "<null>"
```