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 signaturas 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 → bloque
- bloque → { declaraciones instrucciones }

- declaraciones → declaracionesAux
- declaraciones → ε
- declaracionesAux → declaracionesAux declaracion
- declaracionesAux → declaracion
- declaracion → tipo identificador
- declaracion → type tipo identificador
- declaracion → identificador paramsFormales bloque

- paramsFormales → paramsFormalesLista
- paramsFormales → ε

- paramsFormalesLista → paramsFormalesLista param
- paramsFormalesLista → param
- param → tipo & identificador
- param → tipo identificador

- tipo → tipo literalEntero
- tipo → ^ tipo
- tipo → struct listaCampos
- tipo → int
- tipo → real
- tipo → bool
- tipo → string
- tipo → identificador
- listaCampos → listaCampos campo
- listaCampos → campo
- campo → tipo identificador

- instrucciones → instruccionesAux
- instrucciones → ε
- instruccionesAux → instruccionesAux instruccion
- instruccionesAux → instruccion
- instruccion → E
- instruccion → if E bloque
- instruccion → if E bloque else bloque
- instruccion → while E bloque
- instruccion → read E

- instruccion → write E
- instruccion → nl
- instruccion → new E
- instruccion → delete E
- instruccion → identificador paramsReales
- instruccion → bloque
- paramsReales → paramsRealesLista
- paramsReales → ε
- paramsRealesLista → paramsRealesLista E
- paramsRealesLista → E

- E → E = E
- E → E < E
- E → E > E
- E → E <= E
- E → E >= E
- E → E == E
- E → E ! = E
- E → E + E
- E → E − E
- E → E and E
- E → E or E
- E → E * E
- E → E / E
- E → E % E

- E → - E
- E → not E
- E → E [E]
- E → E . identificador
- E → E ^
- E → literalEntero
- E → literalReal
- E → true
- E → false
- E → literalCadena
- E → identificador
- E → 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 construidos arriba, se especifican los constructores.

REGLA	**CONSTRUCTOR**
programa → bloque	prog: Bloq → Prog
bloque → declaraciones instrucciones	bloq: Decs x Insts → Prog
declaraciones → declaracionesAux	si_decs: DecsAux → Decs
declaraciones → ε	no_decs: → Decs
declaracionesAux → declaracionesAux declaracion	muchas_decs: DecsAux x Dec → DecsAux
declaracionesAux → declaracion	una_dec: Dec → DecsAux
declaracion → tipo identificador	dec_var: T x string → Dec
declaracion → type tipo identificador	dec_tipo: T x string → Dec
declaracion → identificador paramsFormales bloque	dec_proc: string x ParamsF x Bloq → Dec
paramsFormales → paramsFormalesLista	si_paramF: ParamsFL → ParamsF
paramsFormales → ε	no_paramF: → ParamsF
paramsFormalesLista → paramsFormalesLista param	muchos_paramsF: ParamsFL x Param → ParamsFL
paramsFormalesLista → param	un_paramF: Param → ParamsFL
param → tipo & identificador	param_ref: T x string → Param
param → tipo identificador	param: T x string → Param
tipo → tipo literalEntero	tipo_array: T x string → T
tipo → ^tipo	tipo_punt: T → T
tipo → struct listaCampos	tipo_struct: LCampos → T
listaCampos → listaCampos campo	muchos_campos: LCampos x Campo → LCampos
listaCampos → campo	un_campo: Campo → LCampos
campo → tipo identificador	campo: T x string → Campo
tipo → int	tipo_int: → T
tipo → real	tipo_real: → T

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

E → E % E	mod: Exp x Exp → Exp
E → - E	negativo: Exp → Exp
E → not E	not: Exp → Exp
E → E [E]	index: Exp x Exp → Exp
E → E . identificador	acceso: Exp x string → Exp
E → E ^	indireccion: Exp → Exp
E → literalEntero	lit_ent: string → Exp
E → literaReal	lit_real: string → Exp
E → true	true: → Exp
E → false	false: → Exp
E → literalCadena	lit_cadena: string → Exp
E → identificador	iden: string → Exp
E → null	null: → 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.

- programa → bloque

  programa.a = prog(bloque a)

- bloque → { declaraciones instrucciones }

  bloque.a = bloq(declaraciones.a, instrucciones.a)

- declaraciones → declaracionesAux &&

  declaraciones.a = si_decs(declaracionesAux.a)

- declaraciones → ε

  declaraciones.a = no_decs()

- declaracionesAux → declaracionesAux ; declaracion

  declaracionesAux1.a = muchas_decs(declaracionesAux2.a, declaracion.a)

- declaracionesAux → declaracion

  declaracionesAux.a = una_dec(declaracion.a)

- declaracion → declaracionVar

  declaracion.a = declaracionVar.a

- declaracion → declaracionTipo

  declaracion = declaracionTipo.a

- declaracion → declaracionProc

  declaracion = declaracionProc.a

- declaracionVar → tipo0 identificador

  declaracionVar.a = dec_var(tipo0.a, identificador.lex)

- declaracionTipo → type tipo0 identificador

  declaracionTipo.a = dec_tipo(tipo0.a, identificador.lex)

- declaracionProc → proc identificador paramsFormales bloque

  declaracionProc.a = dec_proc(identificador.lex, paramsFormales.a, bloque.a)

- paramsFormales → (paramsFormalesAux)

  paramsFormales.a = paramsFormalesAux.a

- paramsFormalesAux → paramsFormalesLista

  paramsFormalesAux.a = si_params(paramsFormalesLista.a)

- paramsFormalesAux → ε

  paramsFormalesAux.a = no_params()

- paramsFormalesLista → paramsFormalesLista , param

  paramsFormalesLista.a = muchos_params(paramsFormalesLista.a, param.a)

- paramsFormalesLista → param

  paramsFormalesLista.a = un_param(param.a)

- param → tipo0 & identificador

  param.a = param_ref(tipo0.a, identificador.lex)

- param → tipo0 identificador
  param.a = param(tipo0.a, identificador.lex)

- tipo0 → tipo0 literalEntero
  $tipo0_1$.a = tipo_array($tipo0_2$.a, literalEntero.lex)

- tipo0 → tipo1
  tipo0.a = tipo.a

- tipo1 → ^ tipo1
  $tipo1_1$.a = tipo_punt($tipo1_2$.a)

- tipo1 → tipo2
  tipo1.a = tipo2.a

- tipo2 → struct { listaCampos }
  tipo2.a = tipo_struct(listaCampos.a)

- listaCampos → listaCampos , campo
  listaCampos1.a = muchos_campos(listaCampos2.a, campo.a)

- listaCampos → campo
  listaCampos.a = un_campo(campo.a)

- campo → tipo0 identificador
  campo.a = campo(tipo0.a, identificador.lex)

- tipo2 → int
  tipo2.a = tipo_int()

- tipo2 → real
  tipo2.a = tipo_real()

- tipo2 → bool
  tipo2.a = tipo_bool()

- tipo2 → string
  tipo2.a = tipo_string()

- tipo2 → identificador

  tipo2.a = tipo_iden(string.lex)

- instrucciones → instruccionesAux

  instrucciones.a = si_instr(instruccionesAux.a)

- instrucciones → ε

  instrucciones.a = no_instr()

- instruccionesAux → instruccionesAux ; instruccion

  instruccionesAux1.a = muchas_instr(instruccionesAux2.a, instruccion.a)

- instruccionesAux → instruccion

  instruccionesAux.a = una_instr(instruccion.a)

- instruccion → @ expr

  instruccion.a = instr_expr(expr.a)

- instruccion → if expr bloque

  instruccion.a = instr_if(expr.a, bloque.a)

- instruccion → if expr bloque else bloque

  instruccion.a = instr_if_else(expr.a, bloque1.a, bloque2.a)

- instruccion → while expr bloque

  instruccion.a = instr_while(expr.a, bloque.a)

- instruccion → read expr

  instruccion.a = instr_read(expr.a)

- instruccion → write expr

  instruccion.a = instr_write(expr.a)

- instruccion → nl

  instruccion.a = instr_nl()

- instruccion → new expr

  instruccion.a = instr_new(expr.a)

- instruccion → delete expr

  instruccion.a = instr_del(expr.a)

- instruccion → call identificador paramsReales

  instruccion.a = instr_call(identificador.lex, paramsReales.a)

- instruccion → bloque

  instruccion.a = instr_bloque(bloque.a)

- paramsReales → (paramsRealesAux)

  paramsReales.a = paramsRealesAux.a

- paramsRealesAux → paramsRealesLista

  paramsRealesAux.a = si_paramsR(paramsRealesLista.a)

- paramsRealesAux → ε

  paramsRealesAux.a = no_paramsR()

- paramsRealesLista → paramsRealesLista , expr

  paramsRealesLista1.a = muchos_paramsR(paramsRealesLista2.a, expr.a)

- paramsRealesLista → expr

  paramsRealesLista.a = un_paramsR(expr.a)

- expr → e0

  expr.a = e0.a

- e0 → e1 = e0

  $e0_1$.a = asig(e1.a, $e0_2$.a)

- e0 → e1

  e0.a = e1.a

- e1 → e1 op1 e2

  $e1_1$.a = mkop1(op1.op, $e1_2$.a, e2.a)

- e1 → e2

  e1.a = e2.a

- e2 → e2 + e3

  $e2_1$.a = suma($e2_2$.a, e3.a)

- e2 → e3 - e3

  e2.a = resta($e3_1$.a, $e3_2$.a)

- e2 → e3

  e2.a = e3.a

- e3 → e4 and e3

  $e3_1$.a = and(e4.a, $e3_2$.a)

- e3 → e4 or e4

  e3.a = or($e4_1$.a, $e4_2$.a)

- e3 → e4

  e3.a = e4.a

- e4 → e4 op4 e5

  $e4_1$.a = mkop4(op4.op, $e4_2$.a, e5.a)

- e4 → e5

  e4.a = e5.a

- e5 → op5 e5

  $e5_1$.a = mkop5(op5.op, $e5_2$.a)

- e5 → e6

  e5.a = e6.a

- e6 → e6 [expr]

  $e6_1$.a = index($e6_2$.a, expr.a)

- e6 → e6 . identificador

  $e6_1$.a = acceso($e6_2$.a, identificador.lex)

- e6 → e6 ^

  $e6_1$.a = indireccion($e6_2$.a)

- e6 → e7

  e6.a = e7.a

- e7 → (e0)

  e7.a = e0.a

- e7 → literalEntero

  e7.a = lit_ent(literalEntero.lex)

- e7 → literaReal

  e7.a = lit_Real(literalReal.lex)

- e7 → true

  e7.a = lit_true()

- e7 → false

  e7.a = lit_false()

- e7 → literalCadena

  e7.a = lit_cadena(literalCadena.lex)

- e7 → identificador

  e7.a = identificador.lex

- e7 → null

  e7.a = null()

- 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 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)

4 Acondicionamiento para permitir la implementación descendente

- programa → bloque

  programa.a = prog(bloque.a)

- bloque → { declaraciones instrucciones }

  bloque.a = bloq(declaraciones.a, instrucciones.a)

- declaraciones → declaracionesAux &&

  declaraciones.a = si_decs(declaracionesAux.a)

- declaraciones → ε

  declaraciones.a = no_decs()

- declaracionesAux → declaracion recDeclaracion

  recDeclaracion.h = una_dec(declaracion.a)

  declaracionesAux.a = recDeclaracion.a

- recDeclaracion → ; declaracion recDeclaracion

  recDeclaracion2.h = muchas_decs(recDeclaracion1.h, declaraciones.a)

  recDeclaracion1.a = recDeclaracion2.a

- recDeclaracion → ε

  recDeclaracion.a = recDeclaracion.h

- declaracion → declaracionVar

  declaracion.a = declaracionVar.a

- declaracion → declaracionTipo

  declaracion = declaracionTipo.a

- declaracion → declaracionProc

  declaracion = declaracionProc.a

- declaracionVar → tipo0 identificador

  declaracionVar.a = dec_var(tipo0.a, identificador.lex)

- declaracionTipo → type tipo0 identificador

  declaracionTipo.a = dec_tipo(tipo0.a, identificador.lex)

- declaracionProc → proc identificador paramsFormales bloque

  declaracionProc.a = dec_proc(identificador.lex, paramsFormales.a, bloque.a)

- paramsFormales → (paramsFormalesAux)

  paramsFormales.a = paramsFormalesAux.a

- paramsFormalesAux → paramsFormalesLista

  paramsFormalesAux.a = si_params(paramsFormalesLista.a)

- paramsFormalesAux → ε

  paramsFormalesAux.a = no_params()

- paramsFormalesLista → param recParamFormal

  recParamFormal.h = un_param(param.a)

  paramsFormalesLista.a = recParamFormal.a

- recParamFormal → , param recParamFormal

  recParamFormal2.h = muchos_params(recParamFormal1.h, param.a)

  recParamFormal1.a = recParamFormal2.a

- recParamFormal → ε

  recParamFormal.a = recParamFormal.h

- param → tipo0 facParam

  facParam.ht = tipo0.a

  param.a = facParam.a

- facParam → & identificador

  facParam.a = param_ref(facParam.ht, identificador.lex)

- facParam → identificador

  facParam.a = param(facParam.h, identificador.lex)

- tipo0 → tipo1 recArray

  recArray.h = tipo1.a

  tipo0.a = recArray.a

- recArray → [literalEntero] recArray

  recArray2.h = tipo_array(recArray1.h, literalEntero.lex)

  recArray1.a = recArray2.a

- recArray → ε

  recArray.a = recArray.h

- tipo1 → ^ tipo1

  $tipo1_1$.a = tipo_punt($tipo1_2$.a)

- tipo1 → tipo2

  tipo1.a = tipo2.a

- tipo2 → struct { listaCampos }

  tipo2.a = tipo_struct(listaCampos.a)

- listaCampos → campo recCampo

  recCampo.h = un_campo(campo.a)

  listaCampos.a = recCampo.a

- recCampo → , campo recCampo

  recCampo2.h = muchos_campos(recCampo1.h, campo.a)

  recCampo1.a = recCampo2.a

- recCampo → ε

  recCampo.a = recCampo.h

- campo → tipo0 identificador

  campo.a = campo(tipo0.a, identificador.lex)

- tipo2 → int

  tipo2.a = tipo_int()

- tipo2 → real

  tipo2.a = tipo_real()

- tipo2 → bool

  tipo2.a = tipo_bool()

- tipo2 → string

  tipo2.a = tipo_string()

- tipo2 → identificador

  tipo2.a = tipo_iden(string.lex)

- instrucciones → instruccionesAux

  instrucciones.a = si_instr(instruccionesAux.a)

- instrucciones → ε

  instrucciones.a = no_instr()

- instruccionesAux → instruccion recInstruccion

  recInstruccion.h = una_instr(instruccion.a)

  instruccionesAux.a = recInstruccion.a

- recInstruccion → ; instruccion recInstruccion

  recInstruccion2.h = muchas_instr(recInstruccion1.h, intruccion.a)

  recInstruccion1.a = recInstruccion2.a

- recInstruccion → ε

  recInstruccion.a = recInstruccion.h

- instruccion → @ expr

  instruccion.a = instr_expr(expr.a)

- instruccion → if expr bloque facIf

  facIf.h1 = expr.a

  facIf.h2 = bloque.a

  instruccion.a = factIf.a

- facIf → else bloque

  facIf.a = instr_if_else(facIf.h1, facIf.h2, bloque.a)

- facIf → ε

  facIf.a = instr_if(facIf.h1, facIf.h2)

- instruccion → while expr bloque

  instruccion.a = instr_while(expr.a, bloque.a)

- instruccion → read expr

  instruccion.a = instr_read(expr.a)

- instruccion → write expr

  instruccion.a = instr_write(expr.a)

- instruccion → nl

  instruccion.a = instr_nl()

- instruccion → new expr

  instruccion.a = instr_new(expr.a)

- instruccion → delete expr

  instruccion.a = instr_del(expr.a)

- instruccion → call identificador paramsReales

  instruccion.a = instr_call(identificador.lex, paramsReales.a)

- instruccion → bloque

  instruccion.a = instr_bloque(bloque.a)

- paramsReales → (paramsRealesAux)

  paramsReales.a = paramsRealesAux.a

- paramsRealesAux → paramsRealesLista

  paramsRealesAux.a = si_paramsR(paramsRealesLista.a)

- paramsRealesAux → ε

  paramsRealesAux.a = no_paramsR()

- paramsRealesLista → expr recParamReal

  recParamReal.h = un_paramsR(expr.a)

  paramsRealesLista.a = recParamReal.a

- recParamReal → , expr recParamReal
  recParamReal2.h = muchos_paramsR(recParamReal1.h, expr.a)
  recParamReal1.a = recParamReal2.a
- recParamReal → ε
  recParamReal.a = recParamReal.h
- expr → e0
- e0 → e1 facE1
  facE1.h = e1.a
  e0.a = facE1.a
- facE1 → = e0
  $facE1_1$.a = asig($facE1_1$.h, e0.a)
- facE1 → ε
  facE1.a = facE1.h
- e1 → e2 recOp1
  recOp1.h = e2.a
  e1.a = recOp1.a
- recOp1 → op1 e2 recOp1
  $recOp1_2$.h = mkop1(op1.op, $recOp1_1$.h, e2.a)
  $recOp1_1$.a = $recOp1_2$.a
- recOp1 → ε
  recOp1.a = recOp1.h
- e2 → e3 facE3 recSuma
  facE3.h = e3.a
  recSuma.h = facE3.a
  e2.a = recSuma.a
- recSuma → + e3 recSuma
  recSuma2.h = suma(recSuma1.h, e3.a)
  recSuma1.a = recSuma2.a

- recSuma → ε

  recSuma.a = recSuma.h

- facE3 → - e3

  facE3.a = resta(fecE3.h, e3.a)

- facE3 → ε

  facE3.a = facE3.h

- e3 → e4 facE4

  facE4.h = e4.a

  e3.a = facE4.a

- facE4 → and e3

  facE4.a = and(facE4.h, e3.a)

- facE4 → or e4

  facE4.a = or(facE4.h, e3.a)

- facE4 → ε

  facE4.a = facE4.h

- e4 → e5 recOp4

  recOp4.h = e5.a

  r4.a = recOp4.a

- recOp4 → op4 e5 recOp4

  $recOp4_2$.h = mkop4(op4.op, $recOp4_1$.h, e5.a)

  $recOp4_1$.a = $recOp4_2$.a

- recOp4 → ε

  recOp4.a = recOp4.h

- e5 → op5 e5

  $e5_1$.a = mkop5(op5.op, $e5_2$.a)

- e5 → e6

  e5.a = e6.a

- e6 → e7 recOp6

  recOp6.h = e7.a

  e6.a = recOp6.a

- recOp6 → [expr] recOp6

  $recOp6_2$.h = index($recOp6_1$.h, expr.a)

  $recOp6_1$.a = $recOp6_2$.a

- recOp6 → . identificador recOp6

  $recOp6_2$.h = acceso(recOp6.a, identificador.lex)

  $recOp6_1$.a = $recOp6_2$.a

- recOp6 → ^ recOp6

  $recOp6_2$.h = indireccion($recOp6_1$.h)

  $recOp6_1$.a = $recOp6_2$.a

- recOp6 → ε

  recOp6.a = recOp6.h

- e7 → (e0)

  e7.a = e0.a

- e7 → literalEntero

  e7.a = lit_ent(literalEntero.lex)

- e7 → literaReal

  e7.a = lit_real(literalReal.lex)

- e7 → true

  e7.a = lit_true()

- e7 → false

  e7.a = lit_false()

- e7 → literalCadena

  e7.a = lit_cadena(literalCadena.lex)

- e7 → identificador

  e7.a = identificador.lex

- e7 → null
  e7.a = null()

- 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 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(Exp,“-”, 5)
- imprime(not(Exp)):
  imprimeExpUnarioPrefijo(Exp,“not”, 5)
- imprime(index(Exp1, Exp2)):
  imprimeOpnd(Exp1,6)
  print “[“
  imprime(Exp2)
  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
- prioridad(lit_ent(_)): return 7
- prioridad(lit_real(_)): return 7
- prioridad(lit_cadena(_)): return 7
- prioridad(true()): return 7
- prioridad(false()): return 7
- prioridad(iden(_)): return 7
- prioridad(null()): return 7