

PRÁCTICA PL: TERCERA FASE II

Tiny(1)

Integrantes:

David Davó Laviña

Ela Katherine Shepherd Arévalo

Grupo 12

ESPECIFICACIÓN SINTAXIS ABSTRACTA

Géneros: Prog(programa), Exp (expresión), Decs (declaraciones), Dec (declaración), Insts (instrucciones), Inst (instrucción), Tipo (tipo), PFs/PF (parámetros formales), Campos (campos), Campo (campo), PR (parámetros reales), PInst (parte de instrucciones), Bloque (bloque)

prog_sin_decs: Insts \rightarrow Prog
prog_con_decs: Decs \times Insts \rightarrow Prog
dec_una: Dec \rightarrow Decs
decs_muchas: Decs \times Dec \rightarrow Decs
var: Tipo \times string \rightarrow Dec
type: Tipo \times string \rightarrow Dec
proc: string \times PFs \times Bloque \rightarrow Dec
param_f_sin: \rightarrow PFs
param_f_con_una: PF \rightarrow PFs
param_f_con_muchas: PFs \times PF \rightarrow PFs
param_f_ref: Tipo \times string \rightarrow PF
param_f_noref: Tipo \times string \rightarrow PF
tipo_array: String \times Tipo \rightarrow Tipo
tipo_record: Campos \rightarrow Tipo
tipo_pointer: Tipo \rightarrow Tipo
tipo_iden: string \rightarrow Tipo
tipo_int: \rightarrow Tipo
tipo_real: \rightarrow Tipo
tipo_bool: \rightarrow Tipo
tipo_string: \rightarrow Tipo
campos_uno: Campo \rightarrow Campos
campos_muchos: Campos \times Campo \rightarrow Campos
campo: Tipo \times string \rightarrow Campo
inst_una: Inst \rightarrow Insts
insts_muchas: Insts \times Inst \rightarrow Insts
e_igual: Exp \times Exp \rightarrow Inst
if: Exp \times PInst \rightarrow Inst
ifelse: Exp \times PInst \times PInst \rightarrow Inst
while: Exp \times PInst \rightarrow Inst
read: Exp \rightarrow Inst
write: Exp \rightarrow Inst
nl: \rightarrow Inst
new: Exp \rightarrow Inst
delete: Exp \rightarrow Inst
call: string \times PR \rightarrow Inst
bl: Bloque \rightarrow Inst

lista_sin: $\rightarrow \text{PInst}$
lista_con: $\text{Insts} \rightarrow \text{PInst}$
param_r_sin: $\rightarrow \text{PR}$
param_r_con_una: $\text{Exp} \rightarrow \text{PR}$
param_r_con_muchas: $\text{PR} \times \text{Exp} \rightarrow \text{PR}$
bloque_sin: $\rightarrow \text{Bloque}$
bloque_con: $\text{Prog} \rightarrow \text{Bloque}$
entero: $\text{string} \rightarrow \text{Exp}$
real: $\text{string} \rightarrow \text{Exp}$
cadena: $\text{string} \rightarrow \text{Exp}$
verdadero: $\rightarrow \text{Exp}$
falso: $\rightarrow \text{Exp}$
null: $\rightarrow \text{Exp}$
identificador: $\text{string} \rightarrow \text{Exp}$
suma: $\text{Exp} \times \text{Exp} \rightarrow \text{Exp}$
resta: $\text{Exp} \times \text{Exp} \rightarrow \text{Exp}$
and: $\text{Exp} \times \text{Exp} \rightarrow \text{Exp}$
or: $\text{Exp} \times \text{Exp} \rightarrow \text{Exp}$
menor: $\text{Exp} \times \text{Exp} \rightarrow \text{Exp}$
men_ig: $\text{Exp} \times \text{Exp} \rightarrow \text{Exp}$
mayor: $\text{Exp} \times \text{Exp} \rightarrow \text{Exp}$
may_ig: $\text{Exp} \times \text{Exp} \rightarrow \text{Exp}$
igual: $\text{Exp} \times \text{Exp} \rightarrow \text{Exp}$
desigual: $\text{Exp} \times \text{Exp} \rightarrow \text{Exp}$
mul: $\text{Exp} \times \text{Exp} \rightarrow \text{Exp}$
div: $\text{Exp} \times \text{Exp} \rightarrow \text{Exp}$
modulo: $\text{Exp} \times \text{Exp} \rightarrow \text{Exp}$
m_unario: $\text{Exp} \rightarrow \text{Exp}$
not: $\text{Exp} \rightarrow \text{Exp}$
indexacion: $\text{Exp} \times \text{Exp} \rightarrow \text{Exp}$
acc_registro: $\text{Exp} \times \text{string} \rightarrow \text{Exp}$
indireccion: $\text{Exp} \rightarrow \text{Exp}$

CONSTRUCTOR AST (GRAMÁTICA S-ATRIBUIDA)

El atributo “a” sintetizado se refiere al árbol de sintaxis abstracto de ese nodo, “lex” es el atributo léxico de los terminales, y “op” es un atributo que se refiere a un operador

Programa -> Decs Insts

Programa.a = prog(Decs.a, Insts.a)

Decs -> ε

Decs.a = null

Decs -> LDecs &&

Decs.a = LDecs.a

LDecs -> Dec

LDecs.a = dec_una(Dec.a)

LDecs -> LDecs ; Dec

LDecs₀.a = decs_muchas(LDecs₁.a, Dec.a)

Dec -> var TypN Identificador

Dec.a = var(TypN.a, Identificador.lex)

Dec -> type TypN Identificador

Dec.a = type(TypN.a, Identificador.lex)

Dec -> proc Identificador ParamsF Bloq

Dec.a = proc(Identificador.lex, ParamsF.a, Bloq.a)

ParamsF -> (LParamFOpc)

ParamsF.a = LParamFOpc.a

LParamFOpc -> ε

LParamFOpc.a = param_f_sin()

LParamFOpc -> LParamF

LParamFOpc.a = LParamF.a

LParamF -> ParamF

LParamF.a = param_f_con_una(ParamF.a)

LParamF -> LParamF , ParamF

LParamF₀.a = param_f_con_muchas(LParamF₁.a, ParamF.a)

ParamF -> TypN & Identificador

ParamF.a = param_f_ref(TypN.a, Identificador.lex)

ParamF -> TypN Identificador

ParamF.a = param_f_noref(TypN.a, Identificador.lex)

TypN -> BaseType

TypN.a = BaseType.a

TypN -> array [LitEnt] of BaseType

TypN.a = tipo_array(LitEnt.a, BaseType.a)

TypN -> record { LCampos }

TypN.a = tipo_record(LCampos.a)

TypN -> pointer BaseType

TypN.a = tipo_pointer(BaseType.a)

LCampos -> Campo

LCampos.a = campo_uno(Campo.a)

LCampos -> LCampos ; Campo

LCampos.a = campo_muchos(LCampos.a, Campo.a)

Campo -> TypN Identificador

Campo.a = campo(TypN.a, Identificador.lex)

BaseType -> BaseType

BaseType.a = BaseType.a

BaseType -> Identificador

BaseType.a = tipo_iden(Identificador.lex)

BasicType -> int

BasicType.a = tipo_int()

BasicType -> realw

BasicType.a = tipo_real()

BasicType -> bool

BasicType.a = tipo_bool()

BasicType -> string

BasicType.a = tipo_string()

Insts -> Inst

Insts.a = inst_una(Inst.a)

Insts -> Insts ; Inst

Insts₀.a = insts_muchas(Insts₁.a, Inst.a)

Inst -> E0 = E0

Inst.a = e_igual(E0₀.a, E0₁.a)

Inst -> if E0 then LInsts endif

Inst.a = if(E0.a, LInsts.a)

Inst -> if E0 then LInsts else LInsts endif

Inst.a = ifelse(E0.a, LInsts₀.a, LInsts₁.a)

Inst -> while E0 do LInsts endwhile

Inst.a = while(E0.a, LInsts.a)

Inst -> read E0

Inst.a = read(E0.a)

Inst -> write E0

Inst.a = write(E0.a)

Inst-> nl

Inst.a = nl()

Inst -> new E0

Inst.a = new(E0.a)

Inst -> delete E0

Inst.a = delete(E0.a)

Inst -> call Identificador ParamsR

Inst.a = call(Identificador.lex, ParamsR.a)

Inst -> Bloq

Inst.a = bl(Bloq.a)

LInsts $\rightarrow \epsilon$
 LInsts.a = lista_sin()
 LInsts \rightarrow Insts
 LInsts.a = lista_con(Insts.a)
 ParamsR \rightarrow (LParamROpc)
 ParamsR.a = LParamROpc.a
 LParamROpc $\rightarrow \epsilon$
 LParamROpc.a = param_r_sin()
 LParamROpc \rightarrow LParamR
 LParamROpc.a = LParamR.a
 LParamR \rightarrow E0
 LParamR.a = param_r_con_una(E0.a)
LParamR \rightarrow LParamR , E0
 LParamR₀.a = param_r_con_muchas(LParamR₁.a, E0.1)
 Bloq \rightarrow { BloqOpc }
 Bloq.a = BloqOpc.a
 BloqOpc $\rightarrow \epsilon$
 BloqOpc.a = bloque_sin()
 BloqOpc \rightarrow Programa
 BloqOpc.a = bloque_con(Programa.a)
E0 \rightarrow E1 OpIn0AsocD E0
 E0₀.a = opera_dos(OpIn0AsocD.op, E1.a, E0₁.a)
E0 \rightarrow E1 OpIn0NoAsoc E1
 E0.a = opera_dos(OpIn0NoAsoc.op, E1₀.a, E1₁.a)
E0 \rightarrow E1
 E0.a = E1.a
E1 \rightarrow E1 OpIn1AsocI E2
 E1₀.a = opera_dos(OpIn1AsocI.op, E1₁.a, E2.a)
 E1 \rightarrow E2
 E1.a = E2.a
E2 \rightarrow E2 OpIn2AsocI E3
 E2₀.a = opera_dos(OpIn2AsocI.op, E2₁.a, E3.a)
 E2 \rightarrow E3
 E2.a = E3.a
E3 \rightarrow E4 OpIn3NoAsoc E4
 E3.a = opera_dos(OpIn3NoAsoc.op, E4₀.a, E4₁.a)
E3 \rightarrow E4
 E3.a = E4.a
 E4 \rightarrow OpPre4NoAsoc E5
 E4.a = opera_uno(OpPre4NoAsoc.op, E5.a)
 E4 \rightarrow OpPre4Asoc E4
 E4₀.a = opera_uno(OpPre4Asoc.op, E4₁.a)
 E4 \rightarrow E5
 E4.a = E5.a

E5 -> E5 OpPos5Asoc

E5₀.a = opera_opposcincoasoc(OpPos5Asoc.op, OpPos5Asoc.a, OpPos5Asoc.var, E5₁.a)

E5 -> E6

E5.a = E6.a

E6 -> OpPre6Asoc E6

E6₀.a = opera_uno(OpPre6Asoc.op, E6₁.a)

E6 -> E7

E6.a = E7.a

E7 -> (E0)

E7.a = E0.a

E7 -> LitEnt

E7.a = entero(LitEnt.lex)

E7 -> LitReal

E7.a = real(LitReal.lex)

E7 -> LitCad

E7.a = cadena(LitCad.lex)

E7 -> true

E7.a = verdadero()

E7 -> false

E7.a = falso()

E7 -> Identificador

E7.a = identificador(Identificador.lex)

E7 -> null

E7.a = null()

OpIn0AsocD -> +

OpIn0AsocD.op = '+'

OpIn0NoAsoc -> -

OpIn0NoAsoc.op = '-'

OpIn1AsocI -> and

OpIn1AsocI.op = 'and'

OpIn1AsocI -> or

OpIn1AsocI.op = 'or'

OpIn2AsocI -> <

OpIn2AsocI.op = '<'

OpIn2AsocI -> <=

OpIn2AsocI.op = '<='

OpIn2AsocI -> >

OpIn2AsocI.op = '>'

OpIn2AsocI -> >=

OpIn2AsocI.op = '>='

OpIn2AsocI -> ==

OpIn2AsocI.op = '=='

OpIn2AsocI -> !=

```

    OpIn2AsocI.op = '!=',
OpIn3NoAsoc -> *
    OpIn3NoAsoc.op = '*'
OpIn3NoAsoc -> /
    OpIn3NoAsoc.op = '/'
OpIn3NoAsoc -> %
    OpIn3NoAsoc.op = '%'
OpPre4NoAsoc -> -
    OpPre4NoAsoc.op = '-'
OpPre4Asoc -> not
    OpPre4Asoc.op = 'not'
OpPos5Asoc -> [ E0 ]
    OpPos5Asoc.op = 'index'
    OpPos5Asoc.a = E0.a
    OpPos5Asoc.var = null
OpPos5Asoc -> .Identificador
    OpPos5Asoc.op = 'reg'
    OpPos5Asoc.a = null
    OpPos5Asoc.var = Identificador.lex
OpPos5Asoc -> ->Identificador
    OpPos5Asoc.op = 'reg'
    OpPos5Asoc.a = null
    OpPos5Asoc.var = Identificador.lex
OpPre6Asoc -> *
    OpPre6Asoc = '*'

```

Funciones semánticas

```

fun prog(Dec, Ins){
    if (Dec == null) then return prog_sin_decs(Ins)
    else return prog_con_decs(Dec, Ins)
}

```

```

fun opera_uno(Op, Arg){
    switch Op
        case '-': return m_unario(Arg)
        case 'not': return not(Arg)
        case '*': return direccion(Arg)
}

```

```

fun opera_dos(Op, Arg0, Arg1){
    switch Op
        case '+': return suma(Arg0, Arg1)
        case '-': return resta(Arg0, Arg1)
        case 'and': return and(Arg0, Arg1)
}

```



```
case 'or': return or(Arg0, Arg1)
case '<': return menor(Arg0, Arg1)
case '<=': return men_ig(Arg0, Arg1)
case '>': return mayor(Arg0, Arg1)
case '>=': return may_ig(Arg0, Arg1)
case '==': return igual(Arg0, Arg1)
case '!=': return desigual(Arg0, Arg1)
case '*': return mul(Arg0, Arg1)
case '/': return div(Arg0, Arg1)
case '%': return modulo(Arg0, Arg1)
```

```
}
```

```
fun opera_opposcincoasoc(Op, Arg_a, Arg_v ,Arg1){
    switch Op
        case 'index': return indexacion(Arg1, Arg_a)
        case 'reg': return acc_registro(Arg1, Arg_v)
}
```

ACONDICIONAMIENTO

En el paso anterior, las reglas en negrita tienen recursión a izquierdas y las reglas en cursiva tienen un factor común. Añadimos un atributo heredado cuando realizamos las transformaciones.

Programa -> Decs Insts

Programa.a = prog(Decs.a, Insts.a)

Decs -> ϵ

Decs.a = null

Decs -> LDecs &&

Decs.a = LDecs.a

LDecs -> Dec RLDecs

LDecs.a = RLDecs.a

RLDecs.ah = dec_una(Dec.a)

RLDecs -> ; Dec RLDecs

RLDecs₀.a = RLDecs₁.a

RLDecs₁.ah = decs_muchas(RLDecs₀.ah, Dec.a)

RLDecs -> ϵ

RLDecs.a = RLDecs.ah

Dec -> var TypN Identificador

Dec.a = var(TypN.a, Identificador.lex)

Dec -> type TypN Identificador

Dec.a = type(TypN.a, Identificador.lex)

Dec -> proc Identificador ParamsF Bloq

Dec.a = proc(Identificador.lex, ParamsF.a, Bloq.a)

ParamsF -> (LParamFOpc)

ParamsF.a = LParamFOpc.a

LParamFOpc -> ϵ

LParamFOpc.a = param_f_sin()

LParamFOpc -> LParamF

LParamFOpc.a = LParamF.a

LParamF -> ParamF RLParamF

LParamF.a = RLParamF.a

RLParamF.ah = param_f_con_una(ParamF.a)

RLParamF -> , ParamF RLParamF

RLParamF₀.a = RLParamF₁.a

RLParamF₁.ah = param_f_con_muchas(RLParamF₀.ah, ParamF.a)

RLParamF -> ϵ

RLParamF.a = RLParamF.ah

ParamF -> TypN RParamF

RParamF.ah = TypN.a

ParamF.a = RParamF.a

RParamF -> & Identificador

```

    RParamF.a = param_f_ref(RParamF.ah, Identificador.lex)
RParamF -> Identificador
    RParamF.a = param_f_noref(RParamF.ah, Identificador.lex)
TypN -> BaseType
    TypN.a = BaseType.a
TypN -> array [ LitEnt ] of BaseType
    TypN.a = tipo_array(LitEnt.a, BaseType.a)
TypN -> record { LCampos }
    TypN.a = tipo_record(LCampos.a)
TypN -> pointer BaseType
    TypN.a = tipo_pointer(BaseType.a)
LCampos -> Campo RLCampos
    LCampos.a = RLCampos.a
    RLCampos.ah = campo_uno(Campo.a)
RLCampos -> ; Campo RLCampos
    RLCampos0.a = RLCampos1.a
    RLCampos1.ah = campo_muchos(RLCampos0.ah, Campo.a)
RLCampos -> ε
    RLCampos.a = RLCampos.ah
Campo -> TypN Identificador
    Campo.a = campo(TypN.a, Identificador.lex)
BaseType -> BaseType
    BaseType.a = BaseType.a
BaseType -> Identificador
    BaseType.a = tipo_iden(Identificador.lex)
BasicType -> int
    BasicType.a = tipo_int()
BasicType -> realw
    BasicType.a = tipo_real()
BasicType -> bool
    BasicType.a = tipo_bool()
BasicType -> string
    BasicType.a = tipo_string()
Insts -> Inst RInsts
    Insts.a = RInsts.a
    RInsts.ah = inst_una(Inst.a)
RInsts -> ; Inst RInsts
    RInsts0.a = RInsts1.a
    RInsts1.ah = insts_muchas(RInsts0.ah, Inst.a)
RInsts -> ε
    RInsts.a = RInsts.ah
Inst -> E0 = E0
    Inst.a = e_igual(E00.a, E01.a)
Inst -> if E0 then LInsts RInst

```

```

    RInst.ah = LInsts.a
    RInst.ahh = E0.a
    Inst.a = RInst.a
RInst -> endif
    RInst.a = if(RInst.ahh, RInst.ah)
RInst -> else LInsts endif
    RInst.a = ifelse(RInst.ahh, RInst.ah, LInsts.a)
Inst -> while E0 do LInsts endwhile
    Inst.a = while(E0.a, LInsts.a)
Inst -> read E0
    Inst.a = read(E0.a)
Inst -> write E0
    Inst.a = write(E0.a)
Inst-> nl
    Inst.a = nl()
Inst -> new E0
    Inst.a = new(E0.a)
Inst -> delete E0
    Inst.a = delete(E0.a)
Inst -> call Identificador ParamsR
    Inst.a = call(Identificador.lex, ParamsR.a)
Inst -> Bloq
    Inst.a = bl(Bloq.a)
LInsts -> ε
    LInsts.a = lista_sin()
LInsts -> Insts
    LInsts.a = lista_con(Insts.a)
ParamsR -> ( LParamROpc )
    ParamsR.a = LParamROpc.a
LParamROpc -> ε
    LParamROpc.a = param_f_sin()
LParamROpc -> LParamR
    LParamROpc.a = LParamR.a
LParamR -> E0 RLParamR
    LParamR.a = RLParamR.a
    RLParamR.ah = param_r_con_una(E0.a)
RLParamR -> , E0 RLParamR
    RLParamR0.a = RLParamR1.a
    RLParamR1.ah = param_r_con_muchas(RLParamR0.ah, E0.a)
RLParamR -> ε
    RLParamR.a = RLParamR.ah
Bloq -> { BloqOpc }
    Bloq.a = BloqOpc.a
BloqOpc -> ε

```

```

    BloqOpc.a = bloque_sin()
BloqOpc -> Programa
    BloqOpc.a = bloque_con(Programa.a)
E0 -> E1 RE0
    E0.a = RE0.a
    RE0.ah = E1.a
RE0 -> OpIn0AsocD E0
    RE0.a = opera_dos(OpIn0AsocD.op, RE0.ah, E0.a)
RE0 -> OpIn0NoAsoc E1
    RE0.a = opera_dos(OpIn0NoAsoc.op, RE0.ah, E1.a)
RE0 -> ε
    RE0.a = RE0.ah
E1 -> E2 RE1
    E1.a = RE1.a
    RE1.ah = E2.a
RE1 -> OpIn1AsocI E2 RE1
    RE10.a = RE11.a
    RE11.ah = opera_dos(OpIn1AsocI.op, RE10.ah ,E2.a)
RE1 -> ε
    RE1.a = RE1.ah
E2 -> E3 RE2
    E2.a = RE2.a
    RE2.ah = E3.a
RE2 -> OpIn2AsocI E3 RE2
    RE20.a = RE21.a
    RE21.ah = opera_dos(OpIn2AsocI.op, RE20.ah ,E3.a)
RE2 -> ε
    RE2.a = RE2.ah
E3 -> E4 RE3
    E3.a = RE3.a
    RE3.ah = E4.a
RE3 -> OpIn3NoAsoc E4
    RE3.a = opera_dos(OpIn3NoAsoc.op, RE3.ah, E4.a)
RE3 -> ε
    RE3.a = RE3.ah
E4 -> OpPre4NoAsoc E5
    E4.a = opera_uno(OpPre4NoAsoc.op, E5.a)
E4 -> OpPre4Asoc E4
    E40.a = opera_uno(OpPre4Asoc.op, E41.a)
E4 -> E5
    E4.a = E5.a
E5 -> E6 RE5
    E5.a = RE5.a
    RE5.ah = E6.a

```

```

RE5 -> OpPos5Asoc RE5
      RE50.a = RE51.a
      RE51.ah = opera_opposcincoasoc(OpPos5Asoc.op, OpPos5Asoc.a, OpPos5Asoc.var,
RE50.ah)
RE5 -> ε
      RE5.a = RE5.ah
E6 -> OpPre6Asoc E6
      E60.a = opera_uno(OpPre6Asoc.op, E61.a)
E6 -> E7
      E6.a = E7.a
E7 -> ( E0 )
      E7.a = E0.a
E7 -> LitEnt
      E7.a = entero(LitEnt.lex)
E7 -> LitReal
      E7.a = real(LitReal.lex)
E7 -> LitCad
      E7.a = cadena(LitCad.lex)
E7 -> true
      E7.a = verdadero()
E7 -> false
      E7.a = falso()
E7 -> Identificador
      E7.a = identificador(Identificador.lex)
E7 -> null
      E7.a = null()
OpIn0AsocD -> +
      OpIn0AsocD.op = '+'
OpIn0NoAsoc-> -
      OpIn0NoAsoc.op = '-'
OpIn1AsocI -> and
      OpIn1AsocI.op = 'and'
OpIn1AsocI -> or
      OpIn1AsocI.op = 'or'
OpIn2AsocI -> <
      OpIn2AsocI.op = '<'
OpIn2AsocI -> <=
      OpIn2AsocI.op = '<='
OpIn2AsocI -> >
      OpIn2AsocI.op = '>'
OpIn2AsocI -> >=
      OpIn2AsocI.op = '>='
OpIn2AsocI -> ==
      OpIn2AsocI.op = '=='

```

```

OpIn2AsocI -> !=
    OpIn2AsocI.op = '!=’
OpIn3NoAsoc -> *
    OpIn3NoAsoc.op = ‘*’
OpIn3NoAsoc -> /
    OpIn3NoAsoc.op = ‘/’
OpIn3NoAsoc -> %
    OpIn3NoAsoc.op = ‘%’
OpPre4NoAsoc -> -
    OpPre4NoAsoc.op = ‘-’
OpPre4Asoc -> not
    OpPre4Asoc.op = ‘not’
OpPos5Asoc -> [ E0 ]
    OpPos5Asoc.op = ‘index’
    OpPos5Asoc.a = E0.a
    OpPos5Asoc.var = null
OpPos5Asoc -> .Identificador
    OpPos5Asoc.op = ‘reg’
    OpPos5Asoc.a = null
    OpPos5Asoc.var = Identificador.lex
OpPos5Asoc -> ->Identificador
    OpPos5Asoc.op = ‘reg’
    OpPos5Asoc.a = null
    OpPos5Asoc.var = Identificador.lex
OpPre6Asoc -> *
    OpPre6Asoc = ‘*’

```