

Memoria realizada por Paula Rosado Fernández, Pablo Cidoncha Cózar y Jaime Pastrana García.

Índice

Especificación léxica.....	1
Especificación sintáctica.....	3
Sintaxis abstracta.....	5
Especificación del constructor de ASTs.....	7
Acondicionamiento de la especificación del constructor de ASTs.....	7
Procesamientos.....	8
Vinculación.....	8
Comprobación de tipos.....	13
Asignación de espacio.....	21
Repertorio de instrucciones de la máquina-p.....	24
Etiquetado.....	26
Generación de código.....	32

Especificación léxica

- Definiciones auxiliares

Letra \equiv [a-z, A-Z]

Dig \equiv [0-9]

DigPos \equiv [1-9]

PEntera \equiv Signo? (0 | DigPos Dig*)

PDecimal \equiv \.(0 | Dig* DigPos)

PExp \equiv (e | E) PEntera

- Clases léxicas

Id \equiv Letra (Letra | Dig | _)*

//Literales

LEntero \equiv PEntera

LReal \equiv PEntera (PDecimal | PExp | PDecimal PExp)

LCadena \equiv ' [^ ' \b \r \n]* '

//Símbolos de operación y puntuación

Suma \equiv \+

Resta \equiv \-

Multiplica \equiv *

Divide \equiv /

Módulo \equiv %
BLT \equiv <
BGT \equiv >
BLE \equiv <=
BGE \equiv >=
BEQ \equiv ==
BNE \equiv !=
PApertura \equiv \
PCierre \equiv \
PuntoComa \equiv ;
Igual \equiv =
CApertura \equiv \
CCierre \equiv \
Punto \equiv \.
Circunflejo \equiv \^
Coma \equiv \,
DosPuntos \equiv \:

//Palabras reservadas

Int \equiv int
Real \equiv real
Bool \equiv bool
String \equiv string
And \equiv and
Or \equiv or
Not \equiv not
Null \equiv null
True \equiv true
False \equiv false
Proc \equiv proc
If \equiv if
Then \equiv then
Else \equiv else
While \equiv while
Do \equiv do
Seq \equiv seq
Begin \equiv begin
End \equiv end
Record \equiv record
Array \equiv array
Of \equiv of
New \equiv new
Delete \equiv delete
Read \equiv read
Write \equiv write
NI \equiv nl
Var \equiv var
Type \equiv type

- Cadenas ignorables

Separadores \equiv $[\backslash s \backslash b \backslash r \backslash n]$

//s es espacio en blanco

Comentarios \equiv $@ [^\backslash n]^*$

Especificación sintáctica

Prog \rightarrow SDecs SIns **Punto**

SDecs \rightarrow SDecs DecT

SDecs \rightarrow DecT //Redundante

SDecs $\rightarrow \epsilon$

DecT \rightarrow Dec **PuntoComa**

Dec \rightarrow DecVar

DecVar \rightarrow **Var Id DosPuntos** Tipo

Dec \rightarrow DecTipo

DecTipo \rightarrow **Type Id DosPuntos** Tipo

Dec \rightarrow DecProc

DecProc \rightarrow **Proc Id** Params SDecs SIns

Tipo \rightarrow TBasico

TBasico \rightarrow **Int**

TBasico \rightarrow **Real**

TBasico \rightarrow **Bool**

TBasico \rightarrow **String**

Tipo \rightarrow TRef

TRef \rightarrow **Id**

Tipo \rightarrow TArray

TArray \rightarrow **Array** **CApertura** **LEntero** **CCierre** **Of** Tipo

Tipo \rightarrow TRecord

TRecord \rightarrow **Record** Campos **End**

Campos \rightarrow Campos CampoT

Campos \rightarrow CampoT

CampoT \rightarrow Campo **PuntoComa**

Campo \rightarrow **Id DosPuntos** Tipo

Tipo \rightarrow TPuntero

TPuntero \rightarrow **Circunflejo** Tipo

ParamsF \rightarrow **PApertura** LParams **PCierre**

LParams \rightarrow LParams **Coma** ParamF //F de Formal

LParams \rightarrow ParamF

LParams $\rightarrow \epsilon$

ParamF \rightarrow ParamRef

ParamF \rightarrow ParamVal

ParamRef → **Var Id DosPuntos** Tipo

ParamVal → **Id DosPuntos** Tipo

SIns → **Begin** LIns **End**

LIns → LIns Ins

LIns → Ins *//Redundante*

LIns → ϵ

Ins → IAsig

IAsig → E **Igual** E **PuntoComa**

Ins → IfThen

IfThen → **If** E **Then** LIns

Ins → IfThenElse

IfThenElse → **If** E **Then** LIns **Else** LIns **End TO**

Ins → IWhile

IWhile → **While** E **Do** LIns **End TO**

Ins → IRead

IRead → **Read** E **PuntoComa**

Ins → IWrite

IWrite → **Write** E **PuntoComa**

Ins → INI *//Aquí pone I de Instrucción y NI de New line*

INI → **NI** **PuntoComa**

Ins → INew

INew → **New** E **PuntoComa**

Ins → IDelete

IDelete → **Delete** E **PuntoComa**

Ins → ICallProc

ICallProc → E ParamsR

ParamsR → **PApertura** LExpr **PCierre** **PuntoComa**

Ins → IComp

IComp → **Seq** SDecs SIns **TO**

TO → **PuntoComa** *//Terminador Opcional*

TO → ϵ

LExp → LExp E

LExp → E *//Redundante*

LExp → ϵ

E → E1 OpRel E1

E → E1

E1 → E2 **Suma** E2

E1 → E1 **Resta** E2

E1 → E2

E2 → E3 **And** E3

E2 → E3 **Or** E2

E2 → E3

E3 → E3 Op3 E4

E3 → E4

E4 → Op4 E4

E4 → E5

E5 → E5 Op5

E5 → EBasic

EBasic → **LEntero**

EBasic → **LReal**

EBasic → **True**

EBasic → **False**

EBasic → **LCadena**

EBasic → **Id**

EBasic → **Null**

EBasic → (E)

OpRel → **BLT**

OpRel → **BGT**

OpRel → **BLE**

OpRel → **BGE**

OpRel → **BNE**

OpRel → **BEQ**

Op3 → **Multiplica**

Op3 → **Divide**

Op3 → **Módulo**

Op4 → **Menos**

Op4 → **Not**

Op5 → **Index**

Index → **CApertura** E **CCierre**

Op5 → **Acces**

Acces → **Punto Id**

Op5 → **Indir**

Indir → **Circunflejo**

Sintaxis abstracta

prog: LDecs x LIns → Prog

sinDecs: → LDecs //Las listas de declaraciones son siempre opcionales

unaDec: Dec → LDecs //Redundante

muchasDecs: LDecs x Dec \rightarrow LDecs

decVar: string x Tipo \rightarrow Dec

decTipo: string x Tipo \rightarrow Dec

decProc: string x LParams x LDecs x LIns \rightarrow Dec

int: \rightarrow Tipo

real: \rightarrow Tipo

bool: \rightarrow Tipo

string: \rightarrow Tipo

~~id: string \rightarrow Tipo??????~~

ref: string \rightarrow Tipo

array: string x Tipo \rightarrow Tipo

record: Campos \rightarrow Tipo

puntero: Tipo \rightarrow Tipo

unCampo: Campo \rightarrow Campos

muchosCampos: Campos x Campo \rightarrow Campos

campo: string x Tipo \rightarrow Campo

sinParams: \rightarrow LParams

unParam: Param \rightarrow LParams

muchosParams: LParams x Param \rightarrow LParams

paramRef: string x Tipo \rightarrow Param //Parámetro por referencia

paramVal: string x Tipo \rightarrow Param //Parámetro por valor (copia)

sinIns: \rightarrow LIns

unIns: Ins \rightarrow LIns //Redundante

muchasIns: LIns x Ins \rightarrow LIns

asignación: E x E \rightarrow Ins

if-then: E x LIns \rightarrow Ins

if-then-else: E x LIns x LIns \rightarrow Ins

while: E x LIns \rightarrow Ins

read: E \rightarrow Ins

write: E \rightarrow Ins

nl: \rightarrow Ins

new: E \rightarrow Ins

delete: E \rightarrow Ins

callProc: E x LExpr \rightarrow Ins

insCompuesta: LDecs x LIns \rightarrow Ins

sinExpr: \rightarrow LExpr

unaExpr: E \rightarrow LExpr //Redundante

muchasExpr: LExpr x E \rightarrow LExpr

int: string \rightarrow E

real: string \rightarrow E

~~bool: string~~ → E

true: → E

false: → E

cadena: string → E

id: string → E

null: → E

//Nivel 0

blt: E x E → E //Operadores relacionales como en ensamblador (i.e. blt = <)

bgt: E x E → E

ble: E x E → E

bge: E x E → E

beq: E x E → E

bne: E x E → E

//Nivel 1

suma: E x E → E

resta: E x E → E

//Nivel 2

and: E x E → E

or: E x E → E

//Nivel 3

mult: E x E → E

div: E x E → E

mod: E x E → E

//Nivel 4

neg: E → E

not: E → E

//Nivel 5

index: E x E → E

access: E x string → E

indir: E → E //Acceso al valor de un puntero (dereferencia o indirección)

Especificación del constructor de ASTs

Prog → SDecs SIns **Punto**

Prog.a = prog(SDecs.a, SIns.a)

SDecs → SDecs DecT

SDecs₀.a = muchasDecs(SDecs₁.a, DecT.a)

SDecs → DecT //Redundante

SDecs.a = unaDec(DecT.a)

SDecs $\rightarrow \epsilon$

SDecs.a = sinDecs()

DecT \rightarrow Dec **PuntoComa**

DecT.a = Dec.a

Dec \rightarrow DecVar

Dec.a = DecVar.a

DecVar \rightarrow **Var Id DosPuntos** Tipo

DecVar.a = decVar(Id.lex, Tipo.a)

Dec \rightarrow DecTipo

Dec.a = DecTipo.a

DecTipo \rightarrow **Type Id DosPuntos** Tipo

DecTipo.a = decTipo(Id.lex, Tipo.a)

Dec \rightarrow DecProc

Dec.a = DecProc.a

DecProc \rightarrow **Proc Id** Params SDecs SIns

DecProc.a = decProc(Id.lex, Params.a, SDecs.a, SIns.a)

Tipo \rightarrow TBasico

Tipo.a = TBasico.a

TBasico \rightarrow **Int**

TBasico.a = int()

TBasico \rightarrow **Real**

TBasico.a = real()

TBasico \rightarrow **Bool**

TBasico.a = bool()

TBasico \rightarrow **String**

TBasico.a = string()

Tipo \rightarrow TRef

Tipo.a = TRef.a

TRef \rightarrow **Id**

TRef.a = ref(Id.lex)

Tipo \rightarrow TArray

Tipo.a = TArray.a

TArray \rightarrow **Array CApertura LEntero CCierre Of** Tipo

TArray.a = array(LEntero.lex, Tipo.a)

Tipo \rightarrow TRecord

Tipo.a = TRecord.a

TRecord \rightarrow **Record** Campos **End**

TRecord.a = record(Campos.a)

Campos \rightarrow Campos CampoT

Campos.a = muchosCampos(Campos.a, CampoT.a)

Campos \rightarrow CampoT

Campos.a = unCampo(CampoT.a)

CampoT \rightarrow Campo **PuntoComa**

CampoT.a = Campo.a

Campo \rightarrow **Id DosPuntos** Tipo

Campo.a = campo(Id.lex, Tipo.a)

Tipo → TPuntero
Tipo.a = TPuntero.a
TPuntero → **Circunflejo** Tipo
TPuntero.a = puntero(Tipo.a)

ParamsF → **PApertura** LParams **PCierre**
ParamsF.a = LParams.a
LParams → LParams **Coma** ParamF //F de Formal
LParams.a = muchosParams(LParams.a, ParamF.a)
LParams → ParamF
LParams.a = unParam(ParamF.a)
LParams → ε
LParams.a = sinParams()
ParamF → ParamRef
ParamF.a = ParamRef.a
ParamF → ParamVal
ParamF.a = ParamVal.a
ParamRef → **Var Id DosPuntos** Tipo
ParamRef.a = paramRef(Id.lex, Tipo.a)
ParamVal → **Id DosPuntos** Tipo
ParamVal.a = paramVal(Id.lex, Tipo.a)

SIns → **Begin** LIns **End**
SIns.a = LIns.a
LIns → LIns Ins
LIns.a = muchasIns(LIns.a, Ins.a)
LIns → Ins //Redundante
LIns.a = unaIns(Ins.a)
LIns → ε
LIns.a = sinIns()

Ins → IAsig
Ins.a = IAsig.a
IAsig → E **Igual** E **PuntoComa**
IAsig.a = asignación(E.a, E.a)
Ins → IIfThen
Ins.a = IIfThen.a
IIfThen → **If** E **Then** LIns
IIfThen.a = if-then(E.a, LIns.a)
Ins → IIfThenElse
Ins.a = IIfThenElse.a
IIfThenElse → **If** E **Then** LIns **Else** LIns **End TO**
IIfThenElse.a = if-then-else(E.a, LIns₀.a, LIns₁.a)
Ins → IWhile
Ins.a = IWhile.a
IWhile → **While** E **Do** LIns **End TO**
IWhile.a = while(E.a, LIns.a)
Ins → IRead

Ins.a = IRead.a
 IRead → **Read E PuntoComa**
 IRead.a = read(E.a)
 Ins → IWrite
 Ins.a = IWrite.a
 IWrite → **Write E PuntoComa**
 IWrite.a = write(E.a)
 Ins → INI //Aquí pone l de Instrucción y NI de New line
 Ins.a = INI.a
 INI → **NI PuntoComa**
 INI.a = nl()
 Ins → INew
 Ins.a = INew.a
 INew → **New E PuntoComa**
 INew.a = new(E.a)
 Ins → IDelete
 Ins.a = IDelete.a
 IDelete → **Delete E PuntoComa**
 IDelete.a = delete(E.a)
 Ins → ICallProc
 Ins.a = ICallProc.a
 ICallProc → E ParamsR
 ICallProc.a = callProc(E.a, ParamsR.a)
 ParamsR → **PApertura LExpr PCierre PuntoComa**
 ParamsR.a = LExpr.a
 Ins → IComp
 Ins.a = IComp.a
 IComp → **Seq SDecs SIns TO**
 IComp.a = insCompuesta(SDecs.a, SIns.a)

 TO → **PuntoComa** //Terminador Opcional
 TO → ϵ

 LExp → LExp E
 LExp.a = muchasExpr(LExpr.a, E.a)
 LExp → E //Redundante
 LExp.a = unaExpr(E.a)
 LExp → ϵ
 LExp.a = sinExpr()

 E → E1 OpRel E1
 E.a = expBin(E1₀.a, OpRel.op, E1₁.a)
 E → E1
 E.a = E1.a

 E1 → E2 **Suma** E2
 E1.a = suma(E2.a, E2.a)
 E1 → E1 **Resta** E2

$E1_0.a = \text{resta}(E1_1.a, E2.a)$
 $E1 \rightarrow E2$
 $E1.a = E2.a$

$E2 \rightarrow E3 \text{ And } E3$
 $E2.a = \text{and}(E3.a, E3.a)$
 $E2 \rightarrow E3 \text{ Or } E2$
 $E2_0.a = \text{or}(E3.a, E2_1.a)$
 $E2 \rightarrow E3$
 $E2.a = E3.a$

$E3 \rightarrow E3 \text{ Op3 } E4$
 $E3_0.a = \text{expBin}(E3_1.a, \text{Op3.op}, E4.a)$
 $E3 \rightarrow E4$
 $E3.a = E4.a$

$E4 \rightarrow \text{Op4 } E4$
 $E4_0.a = \text{expUnPref}(\text{Op4.op}, E4_1.a)$
 $E4 \rightarrow E5$
 $E4.a = E5.a$

$E5 \rightarrow E5 \text{ Op5}$
 $E5_0.a = \text{expUnPos}(E5_1.a, \text{Op5.op})$
 $E5 \rightarrow \text{EBasic}$
 $E5.a = \text{EBasic.a}$

$\text{EBasic} \rightarrow \text{LEntero}$
 $\text{EBasic.a} = \text{int}(\text{LEntero.lex})$
 $\text{EBasic} \rightarrow \text{LReal}$
 $\text{EBasic.a} = \text{real}(\text{LReal.lex})$
 $\text{EBasic} \rightarrow \text{True}$
 $\text{EBasic.a} = \text{true}()$
 $\text{EBasic} \rightarrow \text{False}$
 $\text{EBasic.a} = \text{false}()$
 $\text{EBasic} \rightarrow \text{LCadena}$
 $\text{EBasic.a} = \text{cadena}(\text{LCadena.lex})$
 $\text{EBasic} \rightarrow \text{Id}$
 $\text{EBasic.a} = \text{id}(\text{Id.lex})$
 $\text{EBasic} \rightarrow \text{Null}$
 $\text{EBasic.a} = \text{null}()$
 $\text{EBasic} \rightarrow (E)$
 $\text{EBasic.a} = E.a$

$\text{OpRel} \rightarrow \text{BLT}$
 $\text{OpRel.op} = "<"$
 $\text{OpRel} \rightarrow \text{BGT}$
 $\text{OpRel.op} = ">"$
 $\text{OpRel} \rightarrow \text{BLE}$

OpRel.op = "<="

OpRel → **BGE**

OpRel.op = ">="

OpRel → **BNE**

OpRel.op = "!="

OpRel → **BEQ**

OpRel.op = "=="

Op3 → **Multiplíca**

Op3.op = "*"

Op3 → **Divide**

Op3.op = "/"

Op3 → **Módulo**

Op3.op = "%"

Op4 → **Menos**

Op4.op = "-"

Op4 → **Not**

Op4.op = "not"

Op5 → **Index**

Op5.op = "[]"

Op5.a = Index.a

Index → **CApertura E CCierre**

Index.a = E.a

Op5 → **Acces**

Op5.op = "."

Op5.lex = Access.lex

Acces → **Punto Id**

Access.lex = Id.lex

Op5 → **Indir**

Op5.op = "^"

Indir → **Circunflejo**

expBin(Arg0, Op, Arg1) = //expBin - OpRel, Op3

```
switch Op
case "<": return blt(Arg0, Arg1)
case ">": return bgt(Arg0, Arg1)
case "<=": return ble(Arg0, Arg1)
case ">=": return bge(Arg0, Arg1)
case "!=": return bne(Arg0, Arg1)
case "==": return beq(Arg0, Arg1)
case "*": return mult(Arg0, Arg1)
case "/": return div(Arg0, Arg1)
case "%": return mod(Arg0, Arg1)
```

expUnPref(Op, Arg) = //expUnPref - Op4

```
switch Op
```

```
case "-": return neg(Arg)
case "not": return not(Arg)
```

```
expUnPos(Arg, Op) = //expUnPos - Op5
switch Op
case "[ ]": return index(Arg, Op.a)
case ".": return index(Arg, Op.lex)
case "^": return index(Arg)
```

Acondicionamiento de la especificación del constructor de ASTs para implementación descendente

Prog → SDecs SIns **Punto**
Prog.a = prog(SDecs.a, SIns.a)

SDecs → RSDecs
RSDecs.ah = sinDecs()
SDecs.a = RSDecs.a
RSDecs → DecT RSDecs
RSDecs₁.ah = muchasDecs(RSDecs₀.ah, DecT.a)
RSDecs₀.a = RSDecs₁.a

RSDecs → ε
RSDecs.a = RSDecs.ah

DecT → Dec **PuntoComa**
DecT.a = Dec.a

Dec → DecVar
Dec.a = DecVar.a
DecVar → **Var Id DosPuntos** Tipo
DecVar.a = decVar(Id.lex, Tipo.a)

Dec → DecTipo
Dec.a = DecTipo.a
DecTipo → **Type Id DosPuntos** Tipo
DecTipo.a = decTipo(Id.lex, Tipo.a)

Dec → DecProc
Dec.a = DecProc.a
DecProc → **Proc Id Params** SDecs SIns
DecProc.a = decProc(Id.lex, Params.a, SDecs.a, SIns.a)

Tipo → TBasico
Tipo.a = TBasico.a
TBasico → **Int**
TBasico.a = int()
TBasico → **Real**
TBasico.a = real()

TBasico → **Bool**
 TBasico.a = bool()
 TBasico → **String**
 TBasico.a = string()
 Tipo → TRef
 Tipo.a = TRef.a
 TRef → **Id**
 TRef.a = ref(Id.lex)
 Tipo → TArray
 Tipo.a = TArray.a
 TArray → **Array CApertura LEntero CCierre Of Tipo**
 TArray.a = array(LEntero.lex, Tipo.a)
 Tipo → TRecord
 Tipo.a = TRecord.a
 TRecord → **Record Campos End**
 TRecord.a = record(Campos.a)
 Campos → CampoT RCampos
 RCampos.ah = unCampo(CampoT.a)
 Campos.a = RCampos.a
 RCampos → CampoT RCampos
 RCampos₁.ah = muchosCampos(RCampos₀.ah, CampoT.a)
 RCampos₀.a = RCampos₁.a
 RCampos → ε
 RCampos.a = RCampos.ah
 CampoT → Campo **PuntoComa**
 CampoT.a = Campo.a
 Campo → **Id DosPuntos** Tipo
 Campo.a = campo(Id.lex, Tipo.a)
 Tipo → TPuntero
 Tipo.a = TPuntero.a
 TPuntero → **Circunflejo** Tipo
 TPuntero.a = puntero(Tipo.a)

ParamsF → **PApertura** LParams **PCierre**
 ParamsF.a = LParams.a
 LParams → ParamF RLParams
 RLParams.ah = unParam(ParamF.a)
 LParams.a = RLParams.a
 RLParams → **Coma** ParamF RLParams
 RLParams₁.ah = muchosParams(RLParams₀.ah, ParamF.a)
 RLParams₀.a = RLParams₁.a
 RLParams → ε
 RLParams.a = RLParams.ah
 LParams → ε
 LParams.a = sinParams()
 ParamF → ParamRef
 ParamF.a = ParamRef.a
 ParamF → ParamVal

ParamF.a = ParamVal.a
 ParamRef → **Var Id DosPuntos** Tipo
 ParamRef.a = paramRef(Id.lex, Tipo.a)
 ParamVal → **Id DosPuntos** Tipo
 ParamVal.a = paramVal(Id.lex, Tipo.a)

SIns → **Begin** LIns **End**

 SIns.a = LIns.a

LIns → Ins RLIns

 RLIns.ah = unalns(Ins.a)

 LIns.a = RLIns.a

RLIns → Ins RLIns

 RLIns₁.ah = muchasIns(RLIns₀.a, Ins.a)

 RLIns₀.a = RLIns₁.a

RLIns → ε

 RLIns.a = RLIns.ah

LIns → ε

 LIns.a = sinIns()

Ins → IAsig

 Ins.a = IAsig.a

IAsig → E **Igual** E **PuntoComa**

 IAsig.a = asignación(E.a, E.a)

Ins → IIfThen

 Ins.a = IIfThen.a

IIfThen → **If** E **Then** LIns

 IIfThen.a = if-then(E.a, LIns.a)

Ins → IIfThenElse

 Ins.a = IIfThenElse.a

IIIfThenElse → **If** E **Then** LIns **Else** LIns **End** TO

 IIIfThenElse.a = if-then-else(E.a, LIns₀.a, LIns₁.a)

Ins → IWhile

 Ins.a = IWhile.a

IWhile → **While** E **Do** LIns **End** TO

 IWhile.a = while(E.a, LIns.a)

Ins → IRead

 Ins.a = IRead.a

IRead → **Read** E **PuntoComa**

 IRead.a = read(E.a)

Ins → IWrite

 Ins.a = IWrite.a

IWrite → **Write** E **PuntoComa**

 IWrite.a = write(E.a)

Ins → INI //Aquí pone I de Instrucción y NI de New line

 Ins.a = INI.a

INI → **NI** **PuntoComa**

 INI.a = nl()

Ins → INew

Ins.a = INew.a
 INew → **New** E **PuntoComa**
 INew.a = new(E.a)
 Ins → IDelete
 Ins.a = IDelete.a
 IDelete → **Delete** E **PuntoComa**
 IDelete.a = delete(E.a)
 Ins → ICallProc
 Ins.a = ICallProc.a
 ICallProc → E ParamsR
 ICallProc.a = callProc(E.a, ParamsR.a)
 ParamsR → **PApertura** LExpr **PCierre** **PuntoComa**
 ParamsR.a = LExpr.a
 Ins → IComp
 Ins.a = IComp.a
 IComp → **Seq** SDecs SIns TO
 IComp.a = insCompuesta(SDecs.a, SIns.a)

 TO → **PuntoComa** //Terminador Opcional
 TO → ε

 LExp → E RLExp
 RLExp.ah = unaExpr(E.a)
 LExp.a = RLExp.a
 RLExp → E RLExp
 RLExp1.ah = muchasExpr(RLExp0.a, E.a)
 RLExp0.a = RLExp1.a
 RLExp → ε
 RLExp.a = RLExp.ah
 LExp → ε
 LExp.a = sinExpr()

 E → E1 RE
 RE.ah = E1.a
 E.a = RE.a
 RE → OpRel E1
 RE.a = expBin(RE.ah, OpRel.op, E1.a)
 RE → ε
 RE.a = RE.ah

 E1 → E2 RE1 RE1'
 RE1.ah = E2.a
 RE1'.ah = RE1.a
 E1.a = RE1'.a
 RE1' → **Resta** E2 RE1'
 RE1'₁.ah = resta(RE1'₀.ah, E2.a)
 RE1'₀.a = RE1'₁.a
 RE1' → ε

$RE1'.a = RE1'.ah$
RE1 → Suma E2
 $RE1.a = suma(RE1.ah, E2.a)$
RE1 → ϵ
 $RE1.a = RE1.ah$

E2 → E3 RE2
 $RE2.ah = E3.a$
 $E2.a = RE2.a$
RE2 → And E3
 $RE2.a = and(RE2.ah, E3.a)$
RE2 → Or E2
 $RE2.a = or(RE2.ah, E2.a)$
RE2 → ϵ
 $RE2.a = RE2.ah$

E3 → E4 RE3
 $RE3.ah = E4.a$
 $E3.a = RE3.a$
RE3 → Op3 E4 RE3
 $RE3_1.ah = expBin(RE3_0.ah, Op3.op, E4.a)$
 $RE3_0.a = RE3_1.a$
RE3 → ϵ
 $RE3.a = RE3.ah$

E4 → Op4 E4
 $E4_0.a = expUnPref(Op4, E4_1.a)$
E4 → E5
 $E4.a = E5.a$

E5 → EBasic RE5
 $RE5.ah = EBasic.a$
 $E5.a = RE5.a$
RE5 → Op5
 $RE5.a = expUnPos(RE5.ah, Op5.op)$
RE5 → ϵ
 $RE5.a = RE5.ah$

EBasic → LEntero
 $EBasic.a = int(LEntero.lex)$
EBasic → LReal
 $EBasic.a = real(LReal.lex)$
EBasic → True
 $EBasic.a = true()$
EBasic → False
 $EBasic.a = false()$
EBasic → LCadena
 $EBasic.a = cadena(LCadena.lex)$

EBasic → **Id**
 EBasic.a = id(Id.lex)
EBasic → **Null**
 EBasic.a = null()
EBasic → **(E)**
 EBasic.a = E.a

OpRel → **BLT**
 OpRel.op = "<"
OpRel → **BGT**
 OpRel.op = ">"
OpRel → **BLE**
 OpRel.op = "<="
OpRel → **BGE**
 OpRel.op = ">="
OpRel → **BNE**
 OpRel.op = "!="
OpRel → **BEQ**
 OpRel.op = "=="

Op3 → **Multiplica**
 Op3.op = "*"
Op3 → **Divide**
 Op3.op = "/"
Op3 → **Módulo**
 Op3.op = "%"

Op4 → **Menos**
 Op4.op = "-"
Op4 → **Not**
 Op4.op = "not"

Op5 → **Index**
 Op5.op = "[]"
 Op5.a = Index.a
 Index → **CApertura E CCierre**
 Index.a = E.a
Op5 → **Acces**
 Op5.op = "."
 Op5.lex = Access.lex
 Acces → **Punto Id**
 Access.lex = Id.lex
Op5 → **Indir**
 Op5.op = "^"
 Indir → **Circunflejo**

expBin(Arg0, Op, Arg1) = //expBin - OpRel, Op3
 switch Op

```

case "<": return blt(Arg0, Arg1)
case ">": return bgt(Arg0, Arg1)
case "<=": return ble(Arg0, Arg1)
case ">=": return bge(Arg0, Arg1)
case "!=": return bne(Arg0, Arg1)
case "==": return beq(Arg0, Arg1)
case "*": return mult(Arg0, Arg1)
case "/": return div(Arg0, Arg1)
case "%": return mod(Arg0, Arg1)

```

```
expUnPref(Op, Arg) = //expUnPref - Op4
```

```

switch Op
case "-": return neg(Arg)
case "not": return not(Arg)

```

```
expUnPos(Arg, Op) = //expUnPos - Op5
```

```

switch Op
case "[]": return index(Arg, Op.a)
case ".": return index(Arg, Op.lex)
case "^": return index(Arg)

```

Procesamientos

Vinculación

```
// $ = nodo actual
```

```
global ts //Tabla de símbolos
```

```
vincula(prog(LDecs, LIns)) =
```

```
ts <- nueva_ts() //Inicializar la tabla de símbolos
```

```
vincula1(LDecs)
```

```
vincula2(LDecs)
```

```
vincula(LIns)
```

```
//DECLARACIONES
```

```
//Primera pasada (se vincula todo excepto los pointer refs)
```

```
vincula1(sinDecs()) = skip //No se hace nada porque no hay declaraciones
```

```
vincula1(unaDec(Dec)) = vincula1(Dec)
```

```
vincula1(muchasDecs(LDecs, Dec) =
```

```
vincula1(LDecs) //Recursividad hasta que LDecs sea sinDecs
```

```
vincula1(Dec)
```

```
vincula1(decVar(string, Tipo)) =
```

```
vincula1(Tipo)
```

```
recolecta(string, $) //Se intenta añadir a la tabla de símbolos
```

```
vincula1(decTipo(string, Tipo)) =
```

```
vincula1(Tipo)
```

```

    recolecta(string, $)
vincula1(decProc(string, LParams, LDecs, LIns)) =
    recolecta(string, $)
    ant_ts <- ts
    crea_ambito(ts)    //ts nueva para este ámbito con referencia a la ts padre
    vincula1(LParams)
    vincula1(LDecs)
    vincula2(LParams)
    vincula2(LDecs)
    vincula(LIns)
    ts <- ant_ts

vincula1(int()) = skip
vincula1(real()) = skip
vincula1(bool()) = skip
vincula1(string()) = skip
vincula1(ref(id)) =    //Uso del identificador sin ser puntero(sólo le afectan las decs previas)
    si existe_id(ts, id) entonces
        $.vinculo = valor_de(ts, id)
    si no
        error

vincula1(array(string, Tipo)) =
    vincula1(Tipo)
vincula1(record(Campos)) =
    vincula1(Campos)
vincula1(puntero(Tipo)) =
    si Tipo != ref(_) entonces
        vincula1(T)

vincula1(unCampo(Campo)) =
    vincula1(Campo)
vincula1(muchosCampos(Campos, Campo) =
    vincula1(Campos)
    vincula1(Campo)
vincula1(campo(string, Tipo) =
    vincula1(Tipo)

vincula1(sinParams()) = skip
vincula1(unParam(Param)) =
    vincula1(Param)
vincula1(muchosParams(LParams, Param)) =
    vincula1(LParams)
    vincula1(Param)
vincula1(paramRef(string, Tipo)) =
    vincula1(Tipo)
    recolecta(string, $)
vincula1(paramVal(string, Tipo)) =
    vincula1(Tipo)

```

```

recolecta(string, $)

recolecta(id, Nodo) =
  si existe_id(ts, id) entonces
    error
  si no
    añade(ts, id, Nodo)

//Segunda pasada (se vinculan sólo los pointer refs)
vincula2(sinDecs()) = skip    //No se hace nada porque no hay declaraciones
vincula2(unaDec(Dec)) = vincula1(Dec)
vincula2(muchasDecs(LDecs, Dec) =
  vincula2(LDecs)    //Recursividad hasta que LDecs sea sinDecs
  vincula2(Dec)

vincula2(decVar(string, Tipo)) =
  vincula2(Tipo)
vincula2(decTipo(string, Tipo)) =
  vincula2(Tipo)
vincula2(decProc(string, LParams, LDecs, LIns)) = skip

vincula2(int()) = skip
vincula2(real()) = skip
vincula2(bool()) = skip
vincula2(string()) = skip
vincula2(ref(id)) = skip
vincula2(array(string, Tipo)) =
  vincula2(Tipo)
vincula2(record(Campos)) =
  vincula2(Campos)
vincula2(puntero(Tipo)) =
  si Tipo == ref(id) entonces
    si existe_id(ts, id) entonces
      $.vinculo = valor_de(ts, id)
    si no
      error
  si no
    vincula2(Tipo)

vincula2(unCampo(Campo)) =
  vincula2(Campo)
vincula2(muchosCampos(Campos, Campo) =
  vincula2(Campos)
  vincula2(Campo)
vincula2(campo(string, Tipo) =
  vincula2(Tipo)

vincula2(sinParams()) = skip

```

```

vincula2(unParam(Param)) =
  vincula2(Param)
vincula2(muchosParams(LParams, Param)) =
  vincula2(LParams)
  vincula2(Param)
vincula2(paramRef(string, Tipo)) =
  vincula2(Tipo)
vincula2(paramVal(string, Tipo)) =
  vincula2(Tipo)

```

//INSTRUCCIONES

```

vincula(sinIns()) = skip
vincula(unalns(Ins)) = vincula(Ins)
vincula(muchasIns(LIns, Ins)) =
  vincula(LIns)
  vincula(Ins)

```

//Posibles instrucciones (individuales)

```

vincula(asignacion(E0, E1)) =
  vincula(E0)
  vincula(E1)
vincula(if-then(E, LIns)) =
  vincula(E)
  vincula(LIns)
vincula(if-then-else(E, LIns0, LIns1) =
  vincula(E)
  vincula(LIns0)
  vincula(LIns1)
vincula(while(E, LIns)) =
  vincula(E)
  vincula(LIns)
vincula(read(E)) = vincula(E)
vincula(write(E)) = vincula(E)
vincula(nl()) = skip
vincula(new(E)) = vincula(E)
vincula(delete(E)) = vincula(E)
vincula(callProc(E, LExpr)) =
  vincula(E)
  vincula(LExpr)
vincula(insCompuesta(LDecs, LIns)) =
  ant_ts <- ts
  crea_ambito(ts)  //ts nueva para este ámbito con referencia a la ts padre
  vincula1(LDecs)
  vincula2(LDecs)
  vincula(LIns)
  ts <- ant_ts

vincula(sinExpr()) = skip

```

```
vincula(unaExpr(E)) = vincula(E)
vincula(muchasExpr(LExpr, E)) =
    vincula(LExpr)
    vincula(E)
```

```
vincula(int(string)) = skip
vincula(real(string)) = skip
vincula(true()) = skip
vincula(false()) = skip
vincula(cadena(string)) = skip
vincula(null()) = skip
vincula(id(string)) =
    si existe_id(ts, id) entonces
        $.vinculo = valor_de(ts, id)
    si no
        error
```

```
vincula(blt(E0, E1)) =
    vincula(E0)
    vincula(E1)
vincula(bgt(E0, E1)) =
    vincula(E0)
    vincula(E1)
vincula(ble(E0, E1)) =
    vincula(E0)
    vincula(E1)
vincula(bge(E0, E1)) =
    vincula(E0)
    vincula(E1)
vincula(beq(E0, E1)) =
    vincula(E0)
    vincula(E1)
vincula(bne(E0, E1)) =
    vincula(E0)
    vincula(E1)
vincula(suma(E0, E1)) =
    vincula(E0)
    vincula(E1)
vincula(resta(E0, E1)) =
    vincula(E0)
    vincula(E1)
vincula(and(E0, E1)) =
    vincula(E0)
    vincula(E1)
vincula(or(E0, E1)) =
    vincula(E0)
    vincula(E1)
vincula(mult(E0, E1)) =
```

```

vincula(E0)
vincula(E1)
vincula(div(E0, E1)) =
    vincula(E0)
    vincula(E1)
vincula(mod(E0, E1)) =
    vincula(E0)
    vincula(E1)
vincula(neg(E)) =
    vincula(E)
vincula(not(E)) =
    vincula(E)
vincula(index(E0, E1)) =
    vincula(E0)
    vincula(E1)
vincula(access(E, string)) =
    vincula(E)
vincula(indir(E)) =
    vincula(E)

```

Comprobación de tipos

```

tipado(prog(LDecs, LIns)) =
    tipado(LDecs)    //Restricciones de pre-tipado
    tipado(LIns)
    $.tipo = ambos_ok(LIns.tipo, LDecs.tipo) //El tipo del programa es el de sus instrucciones
    (ok o error)

```

//DECLARACIONES (Restricciones pre-tipado)

```

tipado(sinDecs()) = $.tipo = ok()
tipado(unaDec(Dec)) =
    tipado(Dec)
    $.tipo = Dec.tipo
tipado(muchasDecs(LDecs, Dec)) =
    tipado(LDecs)
    tipado(Dec)
    $.tipo = ambos_ok(LDecs.tipo, Dec.tipo)

```

```

tipado(decVar(id, T)) =
    tipado(T)
    $.tipo = T.tipo
tipado(decTipo(id, T)) =
    tipado(T)
    $.tipo = T.tipo
tipado(decProc(id, LParams, LDecs, LIns)) =
    tipado(LParams)

```



```

tipado(LDecs)
tipado(LIns)
$.tipo = ambos_ok(ambos_ok(LParams.tipo, LDecs.tipo), LIns.tipo)

//Tipos
tipado(int()) = $.tipo = ok()
tipado(real()) = $.tipo = ok()
tipado(bool()) = $.tipo = ok()
tipado(string()) = $.tipo = ok()
tipado(ref(id)) =
    si $.vinculo == decTipo(id, Tipo) entonces //El id es de un tipo declarado (v.g. tTipo)
        $.tipo = ok()
    si no
        error
        $.tipo = error()
tipado(array(tam, T)) =
    tipado(T)
    si T.tipo == error() entonces
        $.tipo = error()
    si no si tam >= 0 entonces //Comprueba que el array tiene un tamaño negativo
        $.tipo = ok()
    si no
        error
        $.tipo = error()
tipado(record(Campos)) =
    tipado(Campos) //Comprueba que no hay campos duplicados y sus tipos son correctos
    $.tipo = Campos.tipo
tipado(puntero(T)) =
    tipado(T)
    $.tipo = T.tipo //Ok o error
tipado(unCampo(Campo)) =
    tipado(Campo)
    $.tipo = Campo.tipo
tipado(muchosCampos(Campos, Campo)) =
    tipado(Campo)
    si Campo.tipo = ok() entonces
        si !esta_en(Campo.string, Campos) entonces
            tipado(Campos)
            $.tipo = Campos.tipo
        si no
            error
            $.tipo = error()
    si no
        $.tipo = Campo.tipo //error
tipado(Campo(string, T)) =
    tipado(T)
    $.tipo = T.tipo

```

//Parámetros

```
tipado(sinParams()) = $.tipo = ok()
tipado(unParam(Param)) =
  tipado(Param)
  $.tipo = Param.tipo
tipado(muchosParams(LParams, Param) =
  tipado(LParams)
  tipado(Param)
  $.tipo = ambos_ok(LParams.tipo, Param.tipo)
tipado(paramRef(id, T)) =
  tipado(T)
  $.tipo = T.tipo
tipado(paramVal(id, T)) =
  tipado(T)
  $.tipo = T.tipo
```

//INSTRUCCIONES

```
tipado(sinIns()) = $.tipo = ok()
tipado(unIns(Ins)) =
  tipado(Ins)
  $.tipo = Ins.tipo
tipado(muchasIns(LIns, Ins)) =
  tipado(LIns)
  tipado(Ins)
  $.tipo = ambos_ok(LIns.tipo, Ins.tipo)

tipado(asignación(E0, E1)) =
  tipado(E0)
  tipado(E1)
  si son_compatibles(E0.tipo, E1.tipo) y es_designador(E0) entonces
    $.tipo = ok()
  si no
    si E0.tipo != error() y E1.tipo != error entonces
      error
    $.tipo = error()
tipado(if-then(E, LIns)) =
  tipado(E)
  tipado(LIns)
  si son_compatibles(E.tipo, bool) entonces
    $.tipo = LIns.tipo
  si no
    $.tipo = error()
    error
tipado(if-then-else(E, LIns0, LIns1)) =
  tipado(E)
  tipado(LIns0)
  tipado(LIns1)
  si son_compatibles(E.tipo, bool) entonces
```

```

        $.tipo = ambos_ok(LIns0.tipo, LIns1.tipo)
si no
    $.tipo = error()
    error
tipado(while(E, LIns)) =
    tipado(E)
    tipado(LIns)
    si son_compatibles(E.tipo, bool) entonces
        $.tipo = LIns.tipo
    si no
        $.tipo = error()
        error
tipado(read(E)) =
    tipado(E)
    si (ref!(E.tipo) == int o ref!(E.tipo) == real o ref!(E.tipo) == string) y es_designador(E)
entonces
    $.tipo = ok()
    si no
        $.tipo = error()
        error
tipado(write(E)) =
    tipado(E)
    si ref!(E.tipo) == int o ref!(E.tipo) == real o ref!(E.tipo) == bool o ref!(E.tipo) == string_
entonces
    $.tipo = ok()
    si no
        $.tipo = error()
        error
tipado(nl()) = $.tipo = ok()
tipado(new(E)) =
    tipado(E)
    si ref!(E.tipo) == puntero entonces
        $.tipo = ok()
    si no
        $.tipo = error()
        si ref!(E.tipo) != error entonces
            error
tipado(delete(E)) =
    tipado(E)
    si ref!(E.tipo) == puntero entonces
        $.tipo = ok()
    si no
        $.tipo = error()
        si ref!(E.tipo) != error entonces
            error
tipado(callProc(E, LExpr)) =
    tipado(E)
    tipado(LExpr)

```

```

//Existe procedimiento vinculado con mismo nombre y num de params
//Todos los parámetros son compatibles con los de la def
//Los parámetros por variable/referencia son designadores
si E.vinculo == decProc(id, LParams, LDecs, LIns) entonces
    si num_elems(LParams) == num_elems(LExpr) entonces
        $.tipo = check_params(LExpr, LParams)
    si no
        $.tipo = error()
        error
si no
    $.tipo = error()
    error
tipado(insCompuesta(LDecs, LIns)) =
    tipado(LIns)
    $.tipo = LIns.tipo

//son_compatibles
son_compatibles(T1, T2) =
    st = vacio()
    return son_compatibles'(st, T1, T2)
son_compatibles'(st, T1, T2) =
    si (T1, T2) pertenece_a(st) entonces
        return true
    si no
        añadir(st, T1, T2)//Set que guarda los tipos que ya se han comprobado (para evitar
        ciclos)

si T1 == ref(id) entonces
    return son_compatibles'(st, ref!(T1), T2)
si no, si T2 == ref(id) entonces
    return son_compatibles'(st, T1, ref!(T2))
si no, si T1 == int y T2 == int entonces
    return true
si no, si T1 == real y (T2 == real or T2 == int) entonces
    return true
si no, si T1 == bool y T2 == bool entonces
    return true
si no, si T1 == string y T2 == string entonces
    return true
si no, si T1 == array(tam1, Tipo1) y T2 == array(tam2, Tipo2) y tam1 == tam2 y
son_compatibles'(st, Tipo1, Tipo2) entonces
    return true
si no, si T1 == record(Campos1) y T2 == record(Campos2) entonces
    return campos_compatibles(Campos1, Campos2)
si no, si T1 == puntero(Tipo) y T2 == null entonces
    return true

```

si no, si T1 == puntero(Tipo1) y T2 == puntero(Tipo2) entonces

return son_compatibles'(st, Tipo1, Tipo2)

si no

return false

campos_compatibles(unCampo(campo(id1, T1)), unCampo(campo(id2, T2))) =

return son_compatibles(T1, T2)

campos_compatibles(muchosCampos(Campos1, campo(id1, T1)),

muchosCampos(Campos2, campo(id2, T2))) =

return son_compatibles(T1, T2) y campos_compatibles(Campos1, Campos2)

campos_compatibles(muchosCampos(_, _), unCampo(_)) = **return** false

campos_compatibles(unCampo(_), muchosCampos(_, _)) = **return** false

//Expresiones

tipado(int(string)) = \$.tipo = int_

tipado(real(string)) = \$.tipo = real

tipado(true()) = \$.tipo = bool

tipado(false()) = \$.tipo = bool

tipado(cadena(string)) = \$.tipo = string

tipado(id(string)) =

sea \$.vinculo = Dec **en**

si Dec == decVar(_, Tipo) **o** Dec == paramRef(_, Tipo) **o** Dec == paramVal(_, Tipo)

entonces

\$.tipo = Tipo

si no

error

\$.tipo = error()

tipado(null()) = \$.tipo = null

tipado(blt(E0, E1)) = \$.tipo = tip_relacional1(E0, E1)

tipado(bgt(E0, E1)) = \$.tipo = tip_relacional1(E0, E1)

tipado(ble(E0, E1)) = \$.tipo = tip_relacional1(E0, E1)

tipado(bge(E0, E1)) = \$.tipo = tip_relacional1(E0, E1)

tipado(beq(E0, E1)) = \$.tipo = tip_relacional2(E0, E1)

tipado(bne(E0, E1)) = \$.tipo = tip_relacional2(E0, E1)

tipado(suma(E0, E1)) = \$.tipo = tip_arit(E0, E1)

tipado(resta(E0, E1)) = \$.tipo = tip_arit(E0, E1)

tipado(mult(E0, E1)) = \$.tipo = tip_arit(E0, E1)

tipado(div(E0, E1)) = \$.tipo = tip_arit(E0, E1)

tipado(mod(E0, E1)) = \$.tipo = tip_mod(E0, E1)

tipado(and(E0, E1)) = \$.tipo = tip_log(E0, E1)

tipado(or(E0, E1)) = \$.tipo = tip_log(E0, E1)

tipado(not(E)) =

si ref!(E) == bool **entonces**

~~**return** bool~~

\$.tipo = bool

si no

error

```

return error()
$.tipo = error
tipado(neg(E)) =
  tipado(E)
  si ref!(E.tipo) == int entonces
    return int
    $.tipo = int
  si no, si ref!(E.tipo) == real entonces
    return real
    $.tipo = real
  si no
    error
    return error()
    $.tipo = error()
tipado(index(E0, E1)) =
  tipado(E0)
  tipado(E1)
  si ref!(E0) == array(tam, tipo) y ref!(E1) == int entonces
    return array(tam, tipo)
    $.tipo = array(tam, tipo)
  si no
    si (ref!(E0.tipo) != error() y ref!(E0.tipo) != array(_, _)) o
      (ref!(E0.tipo) == array(_, _) y ref!(E1.tipo) != error() y ref!(E1.tipo) != int) entonces
        error
        return error()
        $.tipo = error()
tipado(access(E, c)) =
  tipado(E)
  si ref!(E.tipo) == record(Campos) y esta_en(Campos, c) entonces
    return find_tipo(Campos, c)
    $.tipo = find_tipo(Campos, c)
  si no
    si ref!(E.tipo) != error() entonces
      error
      return error()
      $.tipo = error()
tipado(indir(E)) =
  tipado(E)
  si ref!(E.tipo) == puntero(Tipo) entonces
    return Tipo
    $.tipo = Tipo
  si no
    si ref!(E.tipo) != error() entonces
      error
      return error()
      $.tipo = error()

tip_relacional1(E0, E1) =

```

```

tipado(E0)
tipado(E1)
si (ref!(E0.tipo) == int o ref!(E0.tipo) == real) y (ref!(E1.tipo) == int o ref!(E1.tipo) == real)
entonces
    return bool
si no, si ref!(E0.tipo) == bool y ref!(E1.tipo) == bool entonces
    return bool
si no
    error
    return error()
tip_relacional2(E0, E1) =
    tipado(E0)
    tipado(E1)
    si (ref!(E0.tipo) == puntero o ref!(E0.tipo) == null) y (ref!(E1.tipo) == puntero o
    ref!(E1.tipo) == null) entonces
        return bool
    si no, si (ref!(E0.tipo) == int o ref!(E0.tipo) == real) y (ref!(E1.tipo) == int o ref!(E1.tipo) ==
    real) entonces
        return bool
    si no, si ref!(E0.tipo) == bool y ref!(E1.tipo) == bool entonces
        return bool
    si no
        error
        return error()
tip_arit(E0, E1) =
    tipado(E0)
    tipado(E1)
    si ref!(E0.tipo) == int y ref!(E1.tipo) == int entonces
        return int
    si no, si (ref!(E0.tipo) == int o ref!(E0.tipo) == real) y (ref!(E1.tipo) == int o ref!(E1.tipo) ==
    real) entonces
        return real
    si no
        error
        return error()
tip_mod(E0, E1) =
    tipado(E0)
    tipado(E1)
    si ref!(E0.tipo) == int y ref!(E1.tipo) == int entonces
        return int
    si no
        error
        return error()
tip_log(E0, E1) =
    tipado(E0)
    tipado(E1)
    si ref!(E0.tipo) == bool y ref!(E1.tipo) == bool entonces

```

```

    return bool
si no
    error
    return error()

//Funciones auxiliares
ambos_ok(t0, t1) =
    si t0 == ok() y t1 == ok() entonces
        return ok()
    si no
        return error()
es_designador(E) = //Ver si la expresión es algo a lo que le pueda dar un valor (una variable [id],
una pos de array [index], un subcampo [access] o un acceso a puntero [indir])
    si E == id o E == index o E == access o E == indir entonces
        return true
    si no
        return false
ref!(t) = //Seguir la cadena de vínculos si la hay
    mientras t == ref(id) entonces
        t = t.vinculo.tipo
    return t

check_params(sinExpr(), sinParams()) = return ok()
check_params(unaExpr(E), unParam(Param)) = return check_param(E, Param)
check_params(muchasExpr(LExpr, E), muchosParams(LParam, Param)) =
    return ambos_ok(check_params(LExpr, LParams), check_param(E, Param))

check_param(E, paramRef(id, Tipo)) =
    tipado(E)
    si son_compatibles(E.tipo, Tipo) y es_designador(E) y
        (Tipo != real o (E.tipo == real y Tipo == real)) entonces
        return ok()
    si no
        error
        return error()
check_param(E, paramVal(id, Tipo)) =
    tipado(E)
    si son_compatibles(E.tipo, Tipo) entonces
        return ok()
    si no
        error
        return error()

```

Asignación de espacio


```

global dir = 0           //Contador de dirs
global nivel = 0        //Contador de niveles de anidamiento
asigna_espacio(prog(LDecs, LIns)) =
    asigna_espacio(LDecs)
    asigna_espacio(LIns)    //Para las decs de los bloques de código

asigna_espacio(sinDecs()) = skip()
asigna_espacio(unaDec(Dec)) =
    asigna_espacio(Dec)
asigna_espacio(muchasDecs(LDecs, Dec)) =
    asigna_espacio(LDecs)
    asigna_espacio(Dec)

asigna_espacio(decVar(id, Tipo)) =
    $.dir = dir
    $.nivel = nivel
    asigna_espacio_tipo(Tipo)
    dir = dir + Tipo.tam
asigna_espacio(decTipo(id, Tipo)) =
    asigna_espacio_tipo(Tipo)
asigna_espacio(decProc(id, LParams, LDecs, LIns)) =
    ant_dir = dir
    nivel = nivel+1

    $.nivel = nivel
    dir = 0    //Direcciones relativas para el procedimiento
    asigna_espacio(LParams)
    asigna_espacio(LDecs)
    asigna_espacio(LIns)
    $.tam_datos = dir //El espacio ocupado se obtiene de las dirs ocupadas por el proc

    dir = ant_dir
    nivel = nivel-1

//Params de procedimientos
asigna_espacio(sinParams()) = skip()
asigna_espacio(unParam(Param)) =
    asigna_espacio(Param)
asigna_espacio(muchosParams(LParams, Param)) =
    asigna_espacio(LParams)
    asigna_espacio(Param)
asigna_espacio(paramRef(id, Tipo)) =
    $.dir = dir
    $.nivel = nivel
    asigna_espacio(Tipo)
    dir = dir+1 //1 porque es un puntero
asigna_espacio(paramVal(id, Tipo)) =
    $.dir = dir

```

```
$.nivel = nivel
asigna_espacio(Tipo)
dir = dir + Tipo.tam
```

//Tipos

```
asigna_espacio_tipo(T)
  si T.tam == undefined entonces
    asigna_espacio_tipo1(T)      //1ª pasada: todos obtienen un tam asociado (excepto
    algunos tipos de los punteros)
    asigna_espacio_tipo2(T)      //Segunda pasada para los refs de pointers
```

```
asigna_espacio_tipo1(int()) =
  $.tam = 1
asigna_espacio_tipo1(real()) =
  $.tam = 1
asigna_espacio_tipo1(bool()) =
  $.tam = 1
asigna_espacio_tipo1(string()) =
  $.tam = 1
asigna_espacio_tipo1(ref(_)) =
  asigna_espacio_tipo1($.vinculo)
  sea $.vinculo = decTipo(id, T) en
    $.tam = T.tam
asigna_espacio_tipo1(array(tam, T)) =
  asigna_espacio_tipo1(T)
  $.tam = tam * T.tam
asigna_espacio_tipo1(record(Campos)) =
  $.tam = asigna_desplazamiento(Campos)
asigna_espacio_tipo1(puntero(T)) =
  $.tam = 1
  si T != ref(_) entonces
    asigna_espacio_tipo1(T)
```

```
asigna_desplazamiento(unCampo(Campo)) =
  return asigna_desplazamiento(Campo)
asigna_desplazamiento(muchosCampos(Campos, Campo)) =
  Campo.despl = asigna_desplazamiento(Campos)
  return Campo.despl + asigna_desplazamiento(Campo)
asigna_desplazamiento(Campo(id, T)) =
  asigna_espacio_tipo1(T)
  return T.tam
```

```
asigna_espacio_tipo2(int()) = skip()
asigna_espacio_tipo2(real()) = skip()
asigna_espacio_tipo2(bool()) = skip()
asigna_espacio_tipo2(string()) = skip()
asigna_espacio_tipo2(ref(_)) = skip()
asigna_espacio_tipo2(array(tam, T)) =
```

```

    asigna_espacio_tipo2(T)
asigna_espacio_tipo2(record(Campos)) =
    asignacion_espacio_tipo2(Cs)
asignacion_espacio_tipo2(un_campo(campo(id,T))) =
    asignacion_espacio_tipo2(T)
asignacion_espacio_tipo2(muchos_campos(Cs,campo(id,T))) =
    asignacion_espacio_tipo2(Cs)
    asignacion_espacio_tipo2(T)
asigna_espacio_tipo2(puntero(T)) =
    si T == ref(_) entonces
        sea $.vínculo = decTipo(id, T') en
            asigna_espacio_tipo2(T')
            $.tam = T'.tam
    si no
        asigna_espacio_tipo2(T)

```

//Instrucciones (para decs de bloques)

```

asigna_espacio(sinIns()) = skip()
asigna_espacio(unIns(Ins)) =
    asigna_espacio(Ins)
asigna_espacio(muchasIns(LIns, Ins)) =
    asigna_espacio(LIns)
    asigna_espacio(Ins)

asigna_espacio(asigna(E, E)) = skip()
asigna_espacio(if-then(E, LIns)) =
    asigna_espacio(LIns)
asigna_espacio(if-then-else(E, LIns1, LIns2)) =
    asigna_espacio(LIns1)
    asigna_espacio(LIns2)
asigna_espacio(while(E, LIns)) =
    asigna_espacio(LIns)
asigna_espacio(read(E)) = skip()
asigna_espacio(write(E)) = skip()
asigna_espacio(nl()) = skip()
asigna_espacio(new(E)) = skip()
asigna_espacio(delete(E)) = skip()
asigna_espacio(callProc(E, LExpr)) = skip()
asigna_espacio(insCompuesta(LDecs, LIns)) =
    ant_dir = dir
    asigna_espacio(LDecs)
    asigna_espacio(LIns)
    dir = ant_dir

```

Repertorio de instrucciones de la máquina-p

Instrucciones propias de la máquina-p descritas:

- Movimiento de datos:
 - `apilaint(n)`: sitúa en la cima de la pila el valor n.
 - `apilareal(n)`: sitúa en la cima de la pila el valor n.
 - `apilabool(n)`: sitúa en la cima de la pila el valor n.
 - `apilastring(str)`: sitúa en la cima de la pila el valor n.
 - `read[String, Int, Real]`: lee un valor de consola y lo coloca en la cima de la pila
 - `write`: escribe el valor de la cima de la pila y lo elimina.
 - `nl`: escribe un salto de línea.
- Operaciones:
 - `apilaind`: coloca en la cima de la pila el valor de memoria correspondiente a la celda apuntada por la dir situada en la cima de la pila.
 - `desapilaind`: coloca el valor de la cima de la pila en la celda de memoria apuntada por la dir que se encuentra inmediatamente debajo en la pila.
 - `mueve(n)`: copia n celdas desde la dir del valor de la cima de la pila a las celdas consecutivas a la dirección apuntada por el segundo valor de la pila.

Los siguientes operadores aritméticos (a excepción de mod) tienen la opción de ser int o real. Por tanto, suma se subdividiría por ejemplo en `sumaInt` y `sumaReal`.

- `sum`: opera con los dos elementos superiores de la pila, reemplazándolos por el resultado de la operación.
 - `mul`: opera con los dos elementos superiores de la pila, reemplazándolos por el resultado de la operación.
 - `rest`: opera con los dos elementos superiores de la pila, reemplazándolos por el resultado de la operación.
 - `div`: opera con los dos elementos superiores de la pila, reemplazándolos por el resultado de la operación.
 - `mod`: opera con los dos elementos superiores de la pila, reemplazándolos por el resultado de la operación.
 - `and`: opera con los dos elementos superiores de la pila, reemplazándolos por el resultado de la operación.
 - `or`: opera con los dos elementos superiores de la pila, reemplazándolos por el resultado de la operación.
 - `blt, ble, bgt, bge, beq, bne`: opera la comparación correspondiente y coloca true o false en la cima de la pila
 - `not`: niega el valor booleano de la cima de la pila
 - `neg`: invierte el signo del valor int o real de la cima de la pila
 - ~~`realtoint`: convierte el valor de la cima de la pila de real a entero~~
 - `inttoreal`: convierte el valor de la cima de la pila de int a real.
- Saltos
 - `ira(dir)`: salto incondicional a dir.
 - `irf(dir)`: salto condicional si falso en la cima de la pila.
 - `irv(dir)`: salto condicional si verdadero en la cima de la pila.

- irind: salto incondicional a la dirección que se encuentre en la cima de la pila.
- Mem. dinámica:
 - alloc(n): reserva n celdas y apila la dir de comienzo de la primera celda.
 - dealloc(n): libera las n celdas consecutivas a la dir que se encuentre en la cima de la pila.
- Ejecución de procs:
 - activa(n,t,d): prepara todas las estructuras de la máquina-p para ejecutar un procedimiento anidado.
 - apilad(n): emplaza el valor del display correspondiente al nivel n en la cima de la pila.
 - desapilad(n): emplaza el valor de la cima de la pila en el display del nivel n (y lo retira de la pila)
 - desactiva(n, t): prepara todas las estructuras de la máquina-p para retomar la ejecución del proceso padre al que está en el nivel n y tiene tamaño t.
 - dup: duplica el valor de la cima de la pila
 - stop: finaliza la ejecución del programa

Traducción de instrucciones abstractas a las instrucciones de la máquina-p (sirve para comprender mejor la especificación dada del lenguaje):

- alloc(t) = *alloc(t.tam)*
- dealloc(d, t) = *apilaint(d); dealloc(t.tam)*
- fetch(d) = *apilaint(d); apilaind*
- store(d, v) = *apilaint(d); apilaint(v); desapilaind*
- copy(d, d', t) = *apilaint(d); apilaint(d'); mueve(t.tam)*
- indx(d, i, t) = *apilaint(d); apilaint(i); apilaint(t.tam); mul; suma;*
- acc(d, c, t) = *apilaint(d); apilaint(t.desp[c]); suma; //t.desp[c] busca el campo c en el tipo record y devuelve su atributo despl*
- dir(u) = *apilaint(u.vinculo.dir)*

Etiquetado

```

global etq = 0
global procs = pila_vacia()
etiqueta(prog(LDecs, LIns)) =
  $.ini = etq
  etiqueta(LIns)
  etq = etq+1
  recolecta_procs(LDecs)  //Recolecta los procs en la pila de procs
  while (!es_vacia(procs))
    P = pop(procs)
    etiqueta(P)

//Procesos (declaraciones)

```

```

etiqueta(decProc(string, LParams, LDecs, LIns)) =
    $.ini = etq
    etiqueta(LIns)
    //Código de salida del proc
    etq = etq+2
    recolecta_procs(LDecs)
    $.sig = etq

recolecta_procs(sinDecs()) = skip()
recolecta_procs(unaDec(Dec)) = recolecta_procs(Dec)
recolecta_procs(muchasDecs(LDecs, Dec)) =
    recolecta_procs(LDecs)
    recolecta_procs(Dec)
recolecta_procs(decVar(_, _)) = skip()
recolecta_procs(decTipo(_, _)) = skip()
recolecta_procs(decProc(_, _, _, _)) = push($, procs)

```

//Instrucciones

```

etiqueta(sinIns()) = $.ini = etq
etiqueta(unalns(Ins)) =
    $.ini = etq
    etiqueta(Ins)
    $.sig = etq
etiqueta(muchasIns(LIns, Ins)) =
    $.ini = etq
    etiqueta(LIns)
    etiqueta(Ins)
    $.sig = etq

etiqueta(asignacion(E1, E2)) =
    $.ini = etq
    etiqueta(E1)
    etiqueta(E2)
    si E1.tipo == real y E2.tipo == int entonces
        si es_designador(E2) entonces
            etq = etq+1
            etq = etq+2
        si no
            si es_designador(E2) entonces
                etq = etq+1 //Si tienes una dir en E2
            si no
                etq = etq+1 //Si E2 es un inmediato
    $.sig = etq
etiqueta(if-then(E, LIns)) =
    $.ini = etq
    etiqueta(E)
    si es_designador(E) entonces
        etq = etq+1

```

```

    etq = etq+1
    etiqueta(LIns)
    $.sig = etq
etiqueta(if-then-else(E, LIns1, LIns2)) =
    $.ini = etq
    etiqueta(E)
    si es_designador(E) entonces
        etq = etq+1
    etq = etq+1
    etiqueta(LIns1)
    etq = etq+1
    LIns2.ini = etq
    etiqueta(LIns2)
    $.sig = etq
etiqueta(while(E, LIns)) =
    $.ini = etq
    etiqueta(E)
    si es_designador(E) entonces
        etq = etq+1
    etq = etq+1
    etiqueta(LIns)
    etq = etq+1
    $.sig = etq
etiqueta(read(E)) =
    $.ini = etq
    etiqueta(E) //El resultado se deja en la cima de la pila en realidad
    etq = etq+2
    $.sig = etq
etiqueta(write(E)) =
    $.ini = etq
    etiqueta(E)
    si es_designador(E) entonces
        etq = etq+1 //Si tienes una dir en E
    etq = etq+1
    $.sig = etq
etiqueta(nl()) = -escritura de salto de línea-
    $.ini = etq
    etq++;
    $.sig = etq
etiqueta(new(E)) =
    $.ini = etq
    etiqueta(E)
    etq = etq+2
    $.sig = etq
etiqueta(delete(E)) =
    $.ini = etq
    etiqueta(E)
    $.sig_stop = etq+5

```

```

    etq = etq+6
    $.sig = etq
etiqueta(callProc(E, LExpr)) =
    $.ini = etq
    etq = etq+1
    //Salvaguardar variables ocultas por otras -> no entiendo pq ni cómo
    //Reservar espacio (tipo) para los parámetros por valor en nuevas dirs de memoria
    //Reservar espacio (int) para los parámetros por referencia en nuevas dirs de mem
    sea E.vinculo = decProc(string, LParams, LDecs, LIns)
        etiqueta_params(LParams, LExpr)
    //Ejecutar
    etq = etq+2
    $.sig = etq
etiqueta(insCompuesta(LDecs, LIns)) =
    $.ini = etq
    etiqueta(LIns)
    recolecta_procs(LDecs) //Recolecta los procs en la pila de procs
    $.sig = etq

//Expresiones
etiqueta(sinExpr(LExpr)) =
    $.ini = etq
    $.sig = etq
etiqueta(unaExpr(E)) =
    $.ini = etq
    etiqueta(E)
    $.sig = etq
etiqueta(muchasExpr(LExpr, E)) =
    $.ini = etq
    etiqueta(LExpr)
    etiqueta(E)
    $.sig = etq

//Retornan a través de la pila (como en las subrutinas de ensamblador)
etiqueta(int(n)) =
    $.ini = etq
    etq = etq+1
    $.sig = etq
etiqueta(real(n)) =
    $.ini = etq
    etq = etq+1
    $.sig = etq
etiqueta(true()) =
    $.ini = etq
    etq = etq+1
    $.sig = etq
etiqueta(false()) =
    $.ini = etq

```



```

    etq = etq+1
    $.sig = etq
etiqueta(cadena(str)) =
    $.ini = etq
    etq = etq+1
    $.sig = etq
etiqueta(null()) =
    $.ini = etq
    etq = etq+1
    $.sig = etq
etiqueta(id(string)) =
    $.ini = etq
    si $.vinculo.nivel = 0 entonces //Si no es parámetro formal de un proc
        etq = etq+1
    si no
        etq = etq+3
        si $.vinculo = paramRef(str, T)
            etq = etq+1
    $.sig = etq

```

//Operadores relacionales

```

etiqueta_bin_rel(E1, E2) =
    $.ini = etq;
    etiqueta(E1)
    si es_desig(E1) entonces //Si lo que tengo en la pila es una dir y no un valor
        etq = etq+1
    etiqueta(E2)
    si es_desig(E2) entonces //Si lo que tengo en la pila es una dir y no un valor
        etq = etq+1
    etq = etq+1
    $.sig = etq;
etiqueta(bl(E1, E2)) = etiqueta_bin_rel(E1, E2)
etiqueta(bgt(E1, E2)) = etiqueta_bin_rel(E1, E2)
etiqueta(ble(E1, E2)) = etiqueta_bin_rel(E1, E2)
etiqueta(bge(E1, E2)) = etiqueta_bin_rel(E1, E2)
etiqueta(beq(E1, E2)) = etiqueta_bin_rel(E1, E2)
etiqueta(bne(E1, E2)) = etiqueta_bin_rel(E1, E2)

```

t//Operadores aritméticos binarios

```

etiqueta_bin_arit(E1, E2) =
    $.ini = etq
    etiqueta(E1)
    si es_desig(E1) entonces //Si lo que tengo en la pila es una dir y no un valor
        etq = etq+1
    si $.tipo == real entonces
        si E1.tipo == int entonces
            etq = etq+1
    etiqueta(E2)

```

```

si es_desig(E2) entonces //Si lo que tengo en la pila es una dir y no un valor
    etq = etq+1
si $.tipo == real entonces
    si E2.tipo == int entonces
        etq = etq+1
    etq = etq+1
    $.sig = etq
etiqueta(suma(E1, E2)) =
    etiqueta_bin_arit(E1, E2)
etiqueta(resta(E1, E2)) =
    etiqueta_bin_arit(E1, E2)
etiqueta(mult(E1, E2)) =
    etiqueta_bin_arit(E1, E2)
etiqueta(div(E1, E2)) =
    etiqueta_bin_arit(E1, E2)
etiqueta(mod(E1, E2)) =
    etiqueta_bin(E1, E2)

```

//Operadores lógicos (binarios)

```

etiqueta_bin(E1, E2) =
    $.ini = etq
    etiqueta(E1)
    si es_desig(E1) entonces //Si lo que tengo en la pila es una dir y no un valor
        etq = etq+1
    etiqueta(E2)
    si es_desig(E2) entonces //Si lo que tengo en la pila es una dir y no un valor
        etq = etq+1
    etq = etq+1
    $.sig = etq
etiqueta(and(E1, E2)) =
    etiqueta_bin(E1, E2)
etiqueta(or(E1, E2)) =
    etiqueta_bin(E1, E2)

```

//Operadores unarios (infijos)

```

etiqueta_un(E) =
    $.ini = etq
    etiqueta(E)
    si es_desig(E) entonces
        etq = etq+1
    etq = etq+1
    $.sig = etq

```

```

etiqueta(neg(E)) =
    etiqueta_un(E)
etiqueta(not(E)) =
    etiqueta_un(E)

```

//Operadores unarios (sufijos)

```

etiqueta(index(E1, E2)) =

```

```

$.ini = etq
etiqueta(E1)
etiqueta(E2)
si es_desig(E2) entonces
    etq = etq+1
    etq = etq+3
$.sig = etq
etiqueta(access(E, c)) =
    $.ini = etq
    etiqueta(E)
    etq = etq+2
    $.sig = etq
etiqueta(indir(E)) =
    $.ini = etq
    etiqueta(E)
    $.sig_stop = etq+5
    etq = etq+6
    $.sig = etq

```

//Procesos (params)

```

etiqueta_params(sinParams(), sinExpr()) = skip()
etiqueta_params(unParam(P), unaExpr(E)) = etiqueta_paso(P, E)
etiqueta_params(muchosParams(LParams, P), muchasExpr(LExpr, E)) =
    etiqueta_params(LParams, LExpr)
    etiqueta_paso(P, E)

```

```

etiqueta_paso(P, E) =
    etq = etq+3
    etiqueta(E)
    si es_paramVal(P) entonces
        si P.tipo == real y E.tipo == int entonces
            si es_designador(E) entonces
                etq = etq+1
                etq = etq+2
            si no
                si es_designador(E) entonces
                    etq = etq+1 //Si tienes una dir en E
                si no
                    etq = etq+1 //Si E es un inmediato
    si no
        etq = etq+1

```

Generación de código

```

global procs = pila_vacia()
gen_cod(prog(LDecs, LIns)) =

```

```

gen_cod(LIns)
gen_ins(stop)
recolecta_procs(LDecs)  //Recolecta los procs en la pila de procs
while (!es_vacia(procs))
    P = pop(procs)
    gen_cod(P)

//Procesos (declaraciones)
gen_cod(decProc(string, LParams, LDecs, LIns)) =
    gen_cod(LIns)
    //Código de salida del proc
    gen_ins(desactiva($.nivel, $.tam_datos))
    gen_ins(irind())
    recolecta_procs(LDecs)

recolecta_procs(sinDecs()) = skip()
recolecta_procs(unaDec(Dec)) = recolecta_procs(Dec)
recolecta_procs(muchasDecs(LDecs, Dec)) =
    recolecta_procs(LDecs)
    recolecta_procs(Dec)
recolecta_procs(decVar(_, _)) = skip()
recolecta_procs(decTipo(_, _)) = skip()
recolecta_procs(decProc(_, _, _, _)) = push($, procs)

//Instrucciones
gen_cod(sinIns()) = skip()
gen_cod(unalIns(Ins)) = gen_cod(Ins)
gen_cod(muchasIns(LIns, Ins)) =
    gen_cod(LIns)
    gen_cod(Ins)

gen_cod(asignacion(E1, E2)) =
    gen_cod(E1)
    gen_cod(E2)
    si E1.tipo == real y E2.tipo == int entonces
        si es_designador(E2) entonces
            gen_ins(apilaind)
            gen_ins(inttoreal)
            gen_ins(desapilaind)
        si no
            si es_designador(E2) entonces
                gen_ins(mueve(E2.Tipo.tam))  //Si tienes una dir en E2
            si no
                gen_ins(desapilaind)  //Si E2 es un inmediato
gen_cod(if-then(E, LIns)) =
    gen_cod(E)
    si es_designador(E) entonces
        gen_ins(apilaind)

```

```

    gen_ins(irf($.sig))
    gen_cod(LIns)
gen_cod(if-then-else(E, LIns1, LIns2)) =
    gen_cod(E)
    si es_designador(E) entonces
        gen_ins(apilaind)
        gen_ins(irf(LIns2.ini))
        gen_cod(LIns1)
        gen_ins(ir($.sig))
        gen_cod(LIns2)
gen_cod(while(E, LIns)) =
    gen_cod(E)
    si es_designador(E) entonces
        gen_ins(apilaind)
        gen_ins(irf($.sig))
        gen_cod(LIns)
        gen_ins(ir($.ini))
gen_cod(read(E)) =
    gen_cod(E) //El resultado se deja en la cima de la pila en realidad
    -lectura de valor en pila-
    si E.tipo == String entonces
        gen_ins(readString)
    si no si E.tipo == Int entonces
        gen_ins(readInt)
    si no si E.tipo == Real entonces
        gen_ins(readReal)
    gen_ins(desapilaind)
gen_cod(write(E)) =
    gen_cod(E)
    si es_designador(E) entonces
        gen_ins(apilaind) //Si tienes una dir en E
        -escritura del val- gen_ins(write)
gen_cod(nl()) = -escritura de salto de línea- gen_ins(nl)
gen_cod(new(E)) =
    gen_cod(E)
    gen_ins(alloc(E.tipo.tam))
    gen_ins(despilaind)
gen_cod(delete(E)) =
    gen_cod(E)
    gen_ins(dup)
    gen_ins(apilaint(-1))
    gen_ins(beq)
    gen_ins(irf($.sig_stop))
    gen_ins(stop) //(error)
    gen_ins(dealloc(E.tipo.tam))
gen_cod(callProc(E, LExpr)) =
    gen_ins(activa(E.vinculo.nivel, E.vinculo.tam_datos, $.sig/*Volver a justo tras el proc*/))
    //Salvaguardar variables ocultas por otras -> no entiendo pq ni cómo

```

```

//Reservar espacio (tipo) para los parámetros por valor en nuevas dirs de memoria
//Reservar espacio (int) para los parámetros por referencia en nuevas dirs de mem
sea E.vinculo = decProc(string, LParams, LDecs, LIns)
    gen_cod_params(LParams, LExpr)
//Ejecutar
    gen_ins(desapilad(E.vinculo.nivel))
    gen_ins(ira(E.vinculo.ini))
gen_cod(insCompuesta(LDecs, LIns)) =
    gen_cod(LIns)
    recolecta_procs(LDecs) //Recolecta los procs en la pila de procs

//Expresiones
gen_cod(sinExpr(LExpr)) = skip()
gen_cod(unaExpr(E)) = gen_cod(E)
gen_cod(muchasExpr(LExpr, E)) =
    gen_cod(LExpr)
    gen_cod(E)

//Retornan a través de la pila (como en las subrutinas de ensamblador)
gen_cod(int(n)) = gen_ins(apilaint(n))
gen_cod(real(n)) = gen_ins(apilareal(n))
gen_cod(true()) = gen_ins(apilabool(true))
gen_cod(false()) = gen_ins(apilabool(false))
gen_cod(cadena(str)) = gen_ins(apilastring(str))
gen_cod(null()) = gen_ins(apilaint(-1))
gen_cod(id(string)) =
    si $.vinculo.nivel = 0 entonces //Si no es parámetro formal de un proc
        gen_ins(apilaint($.vinculo.dir))
    si no
        gen_ins(apilad($.vinculo.nivel))
        gen_ins(apilaint($.vinculo.dir))
        gen_ins(sumaint())
        si $.vinculo = paramRef(str, T)
            gen_ins(apilaind())

//Operadores relacionales
gen_cod_bin_rel(E1, E2) =
    gen_cod(E1)
    si es_desig(E1) entonces //Si lo que tengo en la pila es una dir y no un valor
        gen_ins(apilaind)
    gen_cod(E2)
    si es_desig(E2) entonces //Si lo que tengo en la pila es una dir y no un valor
        gen_ins(apilaind)
gen_cod(bltn(E1, E2)) =
    gen_cod_bin_rel(E1, E2)
    gen_ins(bltn)
gen_cod(bgt(E1, E2)) =
    gen_cod_bin_rel(E1, E2)

```

```

    gen_ins(bgt)
gen_cod(ble(E1, E2)) =
    gen_cod_bin_rel(E1, E2)
    gen_ins(ble)
gen_cod(bge(E1, E2)) =
    gen_cod_bin_rel(E1, E2)
    gen_ins(bge)
gen_cod(beq(E1, E2)) =
    gen_cod_bin_rel(E1, E2)
    gen_ins(beq)
gen_cod(bne(E1, E2)) =
    gen_cod_bin_rel(E1, E2)
    gen_ins(bne)

```

//Operadores aritméticos binarios

```

gen_cod_bin_arit(E1, E2) =
    gen_cod(E1)
    si es_desig(E1) entonces //Si lo que tengo en la pila es una dir y no un valor
        gen_ins(apilaind)
    si $.tipo == real entonces
        si E1.tipo == int entonces
            gen_ins(inttoreal)
    gen_cod(E2)
    si es_desig(E2) entonces //Si lo que tengo en la pila es una dir y no un valor
        gen_ins(apilaind)
    si $.tipo == real entonces
        si E2.tipo == int entonces
            gen_ins(inttoreal)
gen_cod(suma(E1, E2)) =
    gen_cod_bin_arit(E1, E2)
    si $.tipo == real entonces
        gen_ins(sumaReal)
    si no
        gen_ins(sumaInt)
gen_cod(resta(E1, E2)) =
    gen_cod_bin_arit(E1, E2)
    si $.tipo == real entonces
        gen_ins(restaReal)
    si no
        gen_ins(restaInt)
gen_cod(mult(E1, E2)) =
    gen_cod_bin_arit(E1, E2)
    si $.tipo == real entonces
        gen_ins(mulReal)
    si no
        gen_ins(mullInt)
gen_cod(div(E1, E2)) =
    gen_cod_bin_arit(E1, E2)

```

```

    si $.tipo == real entonces
        gen_ins(divReal)
    si no
        gen_ins(divInt)
gen_cod(mod(E1, E2)) =
    gen_cod_bin(E1, E2)
    gen_ins(mod)

//Operadores lógicos (binarios)
gen_cod_bin(E1, E2) =
    gen_cod(E1)
    si es_desig(E1) entonces
        gen_ins(apilaind)
    gen_cod(E2)
    si es_desig(E2) entonces
        gen_ins(apilaind)
gen_cod(and(E1, E2)) =
    gen_cod_bin(E1, E2)
    gen_ins(and)
gen_cod(or(E1, E2)) =
    gen_cod_bin(E1, E2)
    gen_ins(or)

//Operadores unarios (infijos)
gen_cod_un(E) =
    gen_cod(E)
    si es_desig(E) entonces
        gen_ins(apilaind)
gen_cod(neg(E)) =
    gen_cod_un(E)
    gen_ins(neg)
gen_cod(not(E)) =
    gen_cod_un(E)
    gen_ins(not)

//Operadores unarios (sufijos)
gen_cod(index(E1, E2)) =
    gen_cod(E1)
    gen_cod(E2)
    si es_desig(E2) entonces
        gen_ins(apilaind)
    gen_ins(apilaint(E1.tipo))
    gen_ins(mul)
    gen_ins(suma)
gen_cod(access(E, c)) =
    gen_cod(E)
    gen_ins(apilaint($.tipo.desp[c]))
    gen_ins(suma)
gen_cod(indir(E)) =

```



```

gen_cod(E)
gen_ins(dup)
gen_ins(apilaint(-1))
gen_ins(beq)
gen_ins(irf($.sig_stop))
gen_ins(stop)      //(error)
gen_ins(apilaind)

```

//Procesos (params)

```

gen_cod_params(sinParams(), sinExpr()) = skip()
gen_cod_params(unParam(P), unaExpr(E)) = gen_cod_paso(P, E)
gen_cod_params(muchosParams(LParams, P), muchasExpr(LExpr, E)) =
  gen_cod_params(LParams, LExpr)
  gen_cod_paso(P, E)

```

```

gen_cod_paso(P, E) =
  gen_ins(dup)
  gen_ins(apilaint(P.dir))
  gen_ins(suma)
  gen_cod(E)
  si es_paramVal(P) entonces
    si P.tipo == real y E.tipo == int entonces
      si es_designador(E) entonces
        gen_ins(apilaind)
        gen_ins(inttoreal)
        gen_ins(desapilaind)
      si no
        si es_designador(E) entonces
          gen_ins(mueve(E.Tipo.tam))    //Si tienes una dir en E
        si no
          gen_ins(desapilaind)    //Si E es un inmediato
  si no
    gen_ins(desapila_ind)

```