Jorge Arévalo Echevarría

Ismail Azizi González

Sintaxis Abstracta Tiny.

Especificacion Lexica

- Definiciones Auxiliares:
  - DigPos = [1-9]
  - Dig = (DigPos | 0)
  - Signo =(\+ |\ -)
  - ParteEntera = (0 | DigPos Dig*)
  - ParteDecimal = \. (0 | Dig* DigPos)
  - ParteExponencial = \(e|E) Signo? (ParteEntera)
  - Letra = [a-z] | [A-Z]
  - CaracteresCadena = todos, menos(' \b ' | ' \r ' | ' \n ')
  - ParteCadena = (' CaracteresCadena* ')
- Clases Lexicas
  - Id = Letra(Letra | Dig | _)*
  - Enteros = Signo? ParteEntera
  - Reales = Signo? ParteEntera (ParteDecimal | ParteExponencial | ParteDecimal ParteExponencial)
  - Cadena = ParteCadena
  - Suma = \+
  - Resta = \-
  - Mul = *
  - Div =\ /
  - Mod = \%
  - Menor = \<
  - Mayor = \>
  - MenorIgual = \<=
  - MayorIgual = \>=
  - Igualdad = \==
  - Distinto = \!=
  - ParApertura = \(
  - ParCierre = \)
  - PuntoYComa = \;
  - Igual = \=
  - CorApertura = \[
  - CorCierre =\]
  - Punto = \.
  - DosPuntos = \:
  - Indireccion = \^
  - Exponencial = \E | \e
  - Coma = \,
  - PuntoyComa = \;
- Palabras Reservadas
  - int = (i|I) (n|N) (t|T)
  - real = (r|R) (e|E) (a|A) (l|L)
  - bool = (b|B) (o|O) (o|O) (l|L)
  - string = (s|S) (t|T) (r|R) (i|I) (n|N) (g|G)

    - and = (a|A) (n|N) (d|D)
    - or = (o|O) (r|R)
    - not = (n|N) (o|O) (t|T)
    - null = (n|N) (u|U) (l|L) (l|L)
    - true = (t|T) (r|R) (u|U) (e|E)
    - false = (f|F) (a|A) (l|L) (s|S) (e|E)
    - proc = (p|P) (r|R) (o|O) (c|C)
    - if = (i|I) (f|F)
    - then = (t|T) (h|H) (e|E) (n|N)
    - else = (e|E) (l|L) (s|S) (e|E)
    - while = (w|W) (h|H) (i|I) (l|L) (e|E)
    - do = (d|D) (o|O)
    - seq = (s|S) (e|E) (q|Q)
    - begin = (b|B) (e|E) (g|G) (i|I) (n|N)
    - end = (e|E) (n|N) (d|D)
    - record = (r|R) (e|E) (c|C) (o|O) (r|R) (d|D)
    - array = (a|A) (r|R) (r|R) (a|A) (y|Y)
    - of = (o|O) (f|F)
    - new = (n|N) (e|E) (w|W)
    - delete = (d|D) (e|E) (l|L) (e|E) (t|T) (e|E)
    - read = (r|R) (e|E) (a|A) (d|D)
    - write = (w|W) (r|R) (i|I) (t|T) (e|E)
    - nl = (n|N) (l|L)
    - var = (v|V) (a|A) (r|R)
    - type = (t|T) (y|Y) (p|P) (e|E)

- Cadenas Ignorables
    - Sep = [\b\r\n\(espacio blanco)]
    - Com =@[carcacteres\n]*

<u>Especificacion Sintactica</u>

Prog → Decs SecIs.

Decs → LDecs

Decs → ε

LDecs → LDecs Dec

LDecs → Dec

Dec → DecVar

Dec → DecTipo

Dec → DecProc

DecVar → **var** Id : Tipo ;

DecTipo → **type** Id : Tipo ;

DecProc → **proc** Id (PForms) Decs SecIs

PForms → LPForms

PForms → ε

LPForms → LPForms , ParamFormal

LPForms → ParamFormal

ParamFormal → **var** Id : Tipo

ParamFormal → Id : Tipo

Tipo → TipoBase

Tipo → TipoArray

Tipo → TipoReg

Tipo → TipoPunt

TipoBas → **int** | **real** | **bool** | **string**

TipoArray → **array** [Enteros] **of** TipoBase

TipoReg → **record** Campos **end**

TipoPunt → **^** TipoBase

Campos → Campos Campo

Campos → Campo

Campo → id : Tipo ;

SecIs → **begin** Is **end**

Is → LIs

Is → ε

LIs → LIs I

LIs → I

I → IAsigOLlamada

I → IAlternativa // va comprender las sentencias if

I →Iwhile

I → Iread

I → Iwrite

I → Inl

I → Inew

I → Idelete

I → Iseq

TO → ; // Terminacion Opcional " ; " para Instrucciones

TO → ε

IAlternativa → **if** Exp **then** Is **end** TO //if-then

IAlternativa → **if** Exp **then** Is **else** Is **end** TO //if-then-else

Iwhile → **while** Exp **do** Is **end** TO

Iread → **read** Exp ;

Iwrite → **write** Exp ;

Inl → **nl** ;

Inew → **new** Exp ;

Idelete → **delete** Exp ;

IAsigOLlamada → Exp = Exp ; //Asignacion

IAsigOLlamada → Exp (PReales) ; //Llamada

PReales → LPReales

PReales → ε

LPReales → LPReales , Exp

LPReales → Exp

Iseq → **seq** Decs SecIs TO

OpBinRel → Menor | Mayor | MenorIgual | MayorIgual | Igualdad | Distinto//No asocian

OpBinMas → Suma //No asocia

OpBinMenos → Resta //Asocia izq

OpBinAnd → and //No asocia

OpBinOr → or //Asocia der

OpBin3 → Mul | Div | Mod//Asocia izq

OpUn4 → Menos | not //Asocia, prefijos

Exp → E0

E0 → E1 OpBinRel E1 | E1

E1 → E2 OpBinMas E2 | E1 OpBinMenos E2 | E2

E2 → E3 OpBinAnd E3 | E3 OpBinOr E2| E3

E3 → E3 OpBin3 E4 | E4

E4 → OpUn4 E4 | E5

E5 →E5 [E0] | E5.id | E5^ | E6

E6 → Id | Enteros | Reales | Cadena | true | false | null | (E0)

Gramatica de atributos

Prog → Decs SecIs.

Prog.a = Prog(Decs.a, SecIs.Is)

Decs → LDecs

Decs.a = LDecs.a

Decs → ε

Decs.a = decs_ninguna()

LDecs → LDecs Dec

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

LDecs → Dec

LDecs.a = decs_una(Dec.a)

Dec → DecVar

Dec.a = DecVar.a

Dec → DecTipo

Dec.a = DecTipo.a

Dec → DecProc

Dec.a = DecProc.a

DecVar → **var** Id : Tipo ;

DecVar.a = dec_var(Tipo.a, id.lex)

DecTipo → **type** Id : Tipo ;

DecTipo.a = dec_type(Tipo.a, id.lex)

DecProc → **proc** Id (PForms) Decs SecIs

DecProc.a = dec_proc(id.lex, ParamFormales.a , Decs.a , SecIs.a)

PForms → LPForms

PForms.a = LPForms.a

PForms → ε

PForms.a = pf_ninguno()

LPForms → LPForms , ParamFormal

LPForms0.a = pf_muchos(LPForms1.a, ParamFormal.a)

LPForms → ParamFormal

LPForms.a = pf_uno(ParamFormal.a)

ParamFormal → **var** Id : Tipo

ParamFormal.a = p_var(Tipo.a, Id.lex)

ParamFormal → Id : Tipo

ParamFormal.a = p_valor(Tipo.a, Id.lex)

Tipo → TipoBase

Tipo.a = TipoBase.a

Tipo → TipoArray

Tipo.a = TipoArray.a

Tipo → TipoReg

Tipo.a = TipoReg.a

Tipo → TipoPunt

Tipo.a = TipoPunt.a

TipoBas → **int**

TipoBas.a = int()

TipoBas → **real**

TipoBas.a = real()

TipoBas → **bool**

TipoBas.a = bool()

TipoBas → **string**

TipoBas.a = string()

TipoArray → **array** [Enteros] **of** TipoBase

TipoArray.a =array (TipoBase.a, Enteros.a)

TipoReg → **record** Campos **end**

TipoReg.a = record(Campos.a)

TipoPunt → **^** TipoBase

TipoPunt.a = pointer(TipoBase.a)

Campos → Campos Campo

Campos0.a = campos_muchos(Campos1.a, Campo.a)

Campos → Campo

Campos.a = campos_uno(Campo.a)

Campo → id : Tipo ;

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

SecIs → **begin** Is **end**

SecIS.a = Is.a

Is → LIs

Is.a = LIs.a

Is → ε

Is.a = is_ninguna()

LIs → LIs I

LIs0.a = is_muchas(LIs1.a , I.a)

LIs → I

LIs.a = is_una(I.a)

I → IAsigOLlamada

I.a = IAsigOLlamada.a

I → IAlternativa

I.a = IAlternativa.a

I →Iwhile

I.a = Iwhile.a

I → Iread

I.a = Iread.a

I → Iwrite

I.a = Iwrite.a

I → Inl

I.a = Inl.a

I → Inew

I.a = Inew.a

I → Idelete

I.a = Idelete.a

I → Iseq

I.a = Iseq.a

TO → ; // Terminacion Opcional " ; " para Instrucciones

TO.a = to()

TO → ε

TO.a = to_ninguna()

IAlternativa →**if** Exp **then** Is **end** TO //if-then

IAlternativa.a = is_if_then(Exp.a, Is.a)

IAlternativa →**if** Exp **then** Is **else** Is **end** TO //if-then-else

Iif-then-else.a = is_if_then_else(Exp.a, Is0.a, Is1.a)

Iwhile → **while** Exp **do** Is **end**

Iwhile.a = is_while(Exp.a, Is.a)

Iread → **read** Exp

Iread.a = is_read(Exp.a)

Iwrite → **write** Exp

Iwrite.a = is_write(Exp.a)

Inl → **nl**

Inl.a = is_nl()

Inew → **new** Exp

Inew.a = is_new(Exp.a)

Idelete → **delete** Exp

Idelete.a = is_delete(Exp.a)

IAsigOLlamada → Exp = Exp ; //Asignacion

IAsigOLlamada.a = is_asig(Exp0.a, Exp1.a)

IAsigOLlamada → Exp (PReales) ; //Llamada

IAsigOLlamada.a = is_proc(Exp.a, PReales.a)

PReales → LPReales

PRreales.a = LPReales.a

PReales → ε

PReales.a = pr_ninguno()

LPReales → LPReales , Exp

LPReales0.a = pr_muchos(LPReales1.a, Exp.a)

LPReales → Exp

LPReales.a = pr_uno(Exp.a)

Iseq → **seq** Decs SecIs TO

Iseq.a = is_seq(Decs.a,SecIs.a)

OpBinRel → Menor

OpBinRel.op = “<”

OpBinRel → Mayor

OpBinRel.op = “>”

OpBinRel → MenorIgual

OpBinRel.op = “<=”

OpBinRel → MayorIgual

OpBinRel.op = “>=”

OpBinRel → Igualdad

OpBinRel.op = “==”

OpBinRel → Distinto

OpBinRel.op = “!=”

OpBinMas → Suma

OpBinMas.op = “+”

OpBinMenos → Resta

OpBinMenos.op = “-”

OpBinAnd → and

OpBinAnd.op = “and”

OpBinOr → or

OpBinOr.op = “or”

OpBin3 → Mul

OpBin3.op = “*”

OpBin3 → Div

OpBin3.op = “/”

OpBin3 → Mod

OpBin3.op = “%”

OpUn4 → Menos

OpUn4.op = “-”

OpUn4 →not

OpUn4.op = “not”

Exp → E0

Exp.a = E0.a

E0 → E1 OpBin0Rel E1

E0.a = mkexp(OpBinRel.op,E10.a,E11.a)

E0 → E1

E0.a = E1.a

E1 → E2 OpBinMas E2

E1.a = mkexp(OpBinMas.op,E20.a,E21.a)

E1 → E1 OpBinMenos E2

E10.a = mkexp(OpBinMenos.op,E12.a,E2.a)

E1 → E2

E1.a = E2.a

E2 → E3 OpBinAnd E3

E2.a = mkexp(OpBinAnd.op,E30.a,E31.a)

E2 → E3 OpBinOr E2

E20.a = mkexp(OpBinOr.op,E3.a,E21.a)

E2 → E3

E2.a = E3.a

E3 → E3 OpBin3 E4

E30.a = mkexp(OpBin3.op,E31.a,E4.a)

E3 → E4

E3.a = E4.a

E4 → OpUn4 E4

E40.a = mkexp(OpUn4.op,E41.a)

E4 → E5

E4.a = E5.a

E5 → E5 [E0]

E50.a = exp_index(E51.a)

E5 → E5.id

E50.a = exp_acc_reg(E5.id.a)

E5 → E5^

E50.a = exp_indirreccion(E51.a)

E5 → E6

E5.a = E6.a

E6 → Id

E6.a = id(id.lex)

E6 →Enteros

E6.a = enteros(Enteros.lex)

E6 → Reales

E6.a = reales(Reales.lex)

E6 → Cadena

E6.a = cadena(Cadena.lex)

E6 →true

E6.a = true(true.lex)

E6 →false

E6.a = false(false.lex)

E6 →null

E6.a = null()

E6 → (E0)

E6.a = E0.a

```
fun mkexp(op,arg1,arg2){
switch(op):
        case”+”:return exp_suma(arg1,arg2);
        case”-”:return exp_resta(arg1,arg2);
        case”/”:return exp_div(arg1,arg2);
        case”*”:return exp_mul(arg1,arg2);
        case”and”:return exp_and(arg1,arg2);
        case”or”:return exp_or(arg1,arg2);
        case”<=”:return exp_menor_igual(arg1,arg2);
        case”>=”:return exp_mayor_igual(arg1,arg2);
        case”<”:return exp_menor(arg1,arg2);
        case”>”:return exp_mayor(arg1,arg2);
        case”==”:return exp_igualdad(arg1,arg2);
        case”!=”:return exp_distinto(arg1,arg2);
```

```
}

fun mkexp(op,arg1){
switch(op):
        case”>”:return exp_menos(arg1);
        case”not”:return exp_not(arg1);
}
```

Acondicionamiento Revisada

Prog → Decs SecIs.

Prog.a = Prog(Decs.a, SecIs.Is)

Decs → LDecs

Decs.a = LDecs.a

Decs → ε

Decs.a = decs_ninguna()

LDecs → Dec RLDecs//recursion Izquierdas

RLDecs.ah = decs_una(Dec.a)

LDecs.a = RLDecs.a

RLDecs → Dec RLDecs

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

RLDecs0.a = RLDecs1.a

RLDecs → ε

RLDecs.a =RLDecs.ah

Dec → DecVar

Dec.a = DecVar.a

Dec → DecTipo

Dec.a = DecTipo.a

Dec → DecProc

Dec.a = DecProc.a

DecVar → **var** Id : Tipo ;

DecVar.a = dec_var(Tipo.a, id.lex)

DecTipo → **type** Id : Tipo ;

DecTipo.a = dec_type(Tipo.a, id.lex)

DecProc → **proc** Id (PForms) Decs SecIs

DecProc.a = dec_proc(id.lex, ParamFormales.a , Decs.a , SecIs.a)

PForms → LPForms

PForms.a = RLPForms.a

PForms → ε

PForms.a = pf_ninguno()

LPForms → ParamFormal RLPForms //recursion Izquierdas

RLPForms.ah = pf_uno(ParamFormal.a)

LPForms.a = RLPForms.a

RLPForms → , ParamFormal RLPForms

RLPForms1.ah = pf_muchos(RLPForms0.ah, ParamFormal.a)

RLPForms0.a = RLPForms1.a

RLPForms → ε

RLPForms.a = RLPForms.ah

ParamFormal → **var** Id : Tipo

ParamFormal.a = p_var(Tipo.a, Id.lex)

ParamFormal → Id : Tipo

ParamFormal.a = p_valor(Tipo.a, Id.lex)

Tipo → TipoBase

Tipo.a = TipoBase.a

Tipo → TipoArray

Tipo.a = TipoArray.a

Tipo → TipoReg

Tipo.a = TipoReg.a

Tipo → TipoPunt

Tipo.a = TipoPunt.a

TipoBas → **int**

TipoBas.a = int()

TipoBas → **real**

TipoBas.a = real()

TipoBas → **bool**

TipoBas.a = bool()

TipoBas → **string**

TipoBas.a = string()

TipoArray → **array** [Enteros] **of** TipoBase

TipoArray.a =array (TipoBase.a, Enteros.a)

TipoReg → **record** Campos **end**

TipoReg.a = record(Campos.a)

TipoPunt → **^** TipoBase

TipoPunt.a = pointer(TipoBase.a)

Campos → Campo RCampos //recursion Izquierdas

RCampos.ah = campos_uno(Campo.a)

Campos.a = RCampos.a

RCampos → Campo RCampos

RCampos1.ah = campos_muchos(RCampos0.ah, Campo.a)

RCampos0.a = RCampos1.a

RCampos → ε

RCampos.a = RCampos.ah

Campo → id : Tipo ;

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

SecIs → **begin** Is **end**

SecIS.a = Is.a

Is → LIs

Is.a = LIs.a

Is → ε

Is.a = is_ninguna()

LIs → I RLIs //recursion Izquierdas

RLIs.ah = is_una(I.a)

I.a = RLIs.a

RLIs → I RLIs

RLIs1.ah = is_muchas(RLIs0.ah, I.a)

RLIs0.a = RLIs1.a

RLIs → ε

RLIs.a = RLIs.ah

I → IAsigOLlamada

I.a = IAsigOLlamada.a

I → IAlternativa

I.a = IAlternativa.a

I →Iwhile

I.a = Iwhile.a

I → Iread

I.a = Iread.a

I → Iwrite

I.a = Iwrite.a

I → Inl

I.a = Inl.a

I → Inew

I.a = Inew.a

I → Idelete

I.a = Idelete.a

I → Iseq

I.a = Iseq.a

TO → ; // Terminacion Opcional " ; " para Instrucciones

TO.a = to()

TO → ε

TO.a = to_ninguna()

IAlternativa →**if** Exp **then** Is RAlternativa //factor comun

RAlternativa.ah = pair(Exp.a, Is.a)

IAlternativa.a = RAlternativa.a

RAlternativa → **end** TO

RAlternativa.a = is_if_then(RAlternativa.ah.Exp, RAlternativa.ah.Is)

RAlternativa → **else** Is **end** TO

RAlternativa.a = is_if_then_else(RAlternativa.ah.Exp, RAlternativa.ah.Is, Is.a)

RAlternativa → ε

RAlternativa.a = RAlternativa.ah

Iwhile → **while** Exp **do** Is **end**

Iwhile.a = is_while(Exp.a, Is.a)

Iread → **read** Exp

Iread.a = is_read(Exp.a)

Iwrite → **write** Exp

Iwrite.a = is_write(Exp.a)

Inl → **nl**

Inl.a = is_nl()

Inew → **new** Exp

Inew.a = is_new(Exp.a)

Idelete → **delete** Exp

Idelete.a = is_delete(Exp.a)

IAsigOLlamada →Exp RIAsigOLlamada //factor comun

RIAsigOLlamada.ah = Exp.a

IAsigOLlamada.a = RIAsigOLlamada.a

RIAsigOLlamada → = Exp ;

RIAsigOLlamada.a = is_asig(RIAsigOLlamada.ah, Exp1.a)

RIAsigOLlamada →(PReales) ;

RIAsigOLlamada.a = is_proc(RIAsigOLlamada.ah, PReales.a)

RIAsigOLlamada → ε

RIAsigOLlamada.a = RIAsigOLlamada.ah

PReales → LPReales

PRreales.a = LPReales.a

PReales → ε

PReales.a = pr_ninguno()

LPReales → Exp RLPReales //recursion Izquierdas

RLPReales.ah = pr_uno(Exp.a)

LPReales.a = RLPReales.a

RLPReales → , Exp RLPReales

RLPReales1.ah = pr_muchos(RLPReales0.ah, Exp.a)

RLPReales0.a = RLPReales1.a

RLPReales → ε

RLPReales.a = LPReales.ah

Iseq → **seq** Decs SecIs TO

Iseq.a = is_seq(Decs.a,SecIs.a)

OpBinRel → Menor

OpBinRel.op = "<"

OpBinRel → Mayor

OpBinRel.op = ">"

OpBinRel → MenorIgual

OpBinRel.op = "<="

OpBinRel → MayorIgual

OpBinRel.op = ">="

OpBinRel → Igualdad

OpBinRel.op = "=="

OpBinRel → Distinto

OpBinRel.op = "!="

OpBinMas → Suma

OpBinMas.op = "+"

OpBinMenos → Resta

OpBinMenos.op = "-"

OpBinAnd → and

OpBinAnd.op = "and"

OpBinOr → or

OpBinOr.op = "or"

OpBin3 → Mul

OpBin3.op = "*"

OpBin3 → Div

OpBin3.op = "/"

OpBin3 → Mod

OpBin3.op = "%"

OpUn4 → Menos

OpUn4.op = "-"

OpUn4 →not

OpUn4.op = "not"

Exp → E0

Exp.a = E0.a

E0 → E1 RE0 //factor comun

RE0.ah = E1.a

E0.a = RE0.a

RE0 → OpBinRel E1

RE0.a = mkexp(OpBinRel.op,RE0.ah ,E1.a)

RE0 → ε

RE0.a = RE0.ah

E1 → E2 RE1 RE1' //factor comun y //recursion Izquierdas

RE1.ah = E2.a

RE1'.ah = RE1.a

E1.a = RE1'.a

RE1 → OpBinMas E2 //factor comun

RE1.a = mkexp(OpBinMas.op, RE1.ah, E2.a)

RE1 → ε

RE1.a = RE1.ah

RE1' → OpBinMenos E2 RE1' //recursion Izquierdas

RE1'.ah = mkexp(OpBinMenos.op, RE1'0.ah ,E2.a)

RE1'0.a = RE1'1.a

RE1' → ε

RE1'.a = RE1'.ah

E2 → E3 RE2 //factor comun

RE2.ah = E3.a

E2.a = RE2.a

RE2 → OpBinAnd E3

RE2.a =mkexp(OpBinAnd.op, RE2.ah , E3.a)

RE2 → OpBinOr E2

RE2.a = mkexp(OpBinOr.op, RE2.ah, E2.a)

RE2 → ε

RE0.a = RE0.ah

E3 → E4 RE3 //recursion Izquierdas

RE3.ah = E4.a

E3.a = RE3.a

RE3 → OpBin3 E4 RE3

RE31.ah = mkexp(OpBin3.op, RE30.ah, E4.a)

RE30.a = RE31.a

RE3 → ε

RE3.a = RE3.ah

E4 → OpUn4 E4

E40.a = mkexp(OpUn4.op,E41.a)

E4 → E5

E4.a = E5.a

E5 → E6 RE5 //recursion Izquierdas

RE5.ah = E6.a

E5.a = RE5.a

RE5 → [E0] RE5

RE51.ah = exp_index(RE50.ah)

RE50.a = RE51.a

RE5 → .id RE5

RE51.ah = exp_index(RE50.id.ah)

RE50.a = RE51.a

RE5 → ^ RE5

RE51.ah = exp_indirreccion(RE50.ah)

RE50.a = RE51.a

RE5 → ε

RE5.a = RE5.ah

E6 → Id

E6.a = id(id.lex)

E6 →Enteros

E6.a = enteros(Enteros.lex)

E6 → Reales

E6.a = reales(Reales.lex)

E6 → Cadena

E6.a = cadena(Cadena.lex)

E6 →true

E6.a = true(true.lex)

E6 →false

E6.a = false(false.lex)

E6 →null

E6.a = null()

E6 → (E0)

E6.a = E0.a

```
fun mkexp(op,arg1,arg2){
switch(op):
        case”+”:return exp_suma(arg1,arg2);
        case”-”:return exp_resta(arg1,arg2);
        case”/”:return exp_div(arg1,arg2);
        case”*”:return exp_mul(arg1,arg2);
        case”and”:return exp_and(arg1,arg2);
        case”or”:return exp_or(arg1,arg2);
        case”<=”:return exp_menor_igual(arg1,arg2);
        case”>=”:return exp_mayor_igual(arg1,arg2);
        case”<”:return exp_menor(arg1,arg2);
        case”>”:return exp_mayor(arg1,arg2);
        case”==”:return exp_igualdad(arg1,arg2);
        case”!=”:return exp_distinto(arg1,arg2);
}

fun mkexp(op,arg1){
switch(op):
        case”>”:return exp_menos(arg1);
        case”not”:return exp_not(arg1);
}
```