

---

# PROCESADORES DE LENGUAJES

---

MEMORIA DE PROYECTO - HITO 4: COMPILADOR

## GRUPO 10

SERGIO COLET GARCÍA  
LAURA MARTÍNEZ TOMÁS  
RODRIGO SOUTO SANTOS  
LI JIE CHEN CHEN

*GRADO EN INGENIERÍA INFORMÁTICA  
FACULTAD DE INFORMÁTICA  
UNIVERSIDAD COMPLUTENSE DE MADRID*



2023-2024

# Índice general

# 1 | Especificación del procesamiento de vinculación

---

```
var ts //Tabla de simbolos

vincula(prog(Blo)):
    ts = creaTS()
    vincula(Blo)

vincula(bloq(Decs, Insts)):
    abreAmbito(ts)
    vincula(Decs)
    vincula(Insts)
    cierraAmbito(ts)

vincula(si_decs(LDecs)):
    vincula1(LDecs)
    vincula2(LDecs)

vincula1(no_decs()): noop

vincula1(muchas_decs(LDecs, Dec)):
    vincula1(LDecs)
    vincula1(Dec)

vincula2(muchas_decs(LDecs, Dec)):
    vincula2(LDecs)
    vincula2(Dec)

vincula1(una_dec(Dec)):
    vincula1(Dec)

vincula2(una_dec(Dec)):
    vincula2(Dec)

vincula1(muchas_var(LVar, Var)):
    vincula1(LVar)
    vincula1(Var)

vincula2(muchas_var(LVar, Var)):
    vincula2(LVar)
    vincula2(Var)

vincula1(una_var(Var)):
    vincula1(Var)

vincula2(una_var(Var)):
    vincula2(Var)

vincula1(var(Tipo, id)):
    vincula1(Tipo)
    if contiene(ts, id) then
        error
    else
        inserta(ts, id, $)
    end if
```

```

vincula2 (var (Tipo , id )):
    vincula2 (Tipo)

vincula1 (dec_simple (Var )):
    vincula1 (Var)

vincula2 (dec_simple (Var )):
    vincula2 (Var)

vincula1 (dec_type (Var )):
    vincula1 (Var)

vincula2 (dec_type (Var )):
    vincula2 (Var)

vincula1 (dec_proc (id , PFmls , Blo )):
    if contiene (ts , id) then
        error
    else
        inserta (ts , id , $)
    end if

vincula2 (dec_proc (id , PFmls , Blo )):
    abreAmbito (ts)
    vincula2 (PFmls)
    vincula1 (PFmls)
    vincula (Blo)
    cierraAmbito (ts)

vincula1 (tipo_array (Tipo , litEnt )):
    if Tipo != tipo_ident (__) then
        vincula1 (Tipo)
    end if

vincula2 (tipo_array (Tipo , litEnt )):
    if Tipo == tipo_ident (id) then
        Tipo.vinculo = vinculoDe (ts , id)
        if Tipo.vinculo != dec_type (__) then
            error
        end if
    else
        vincula2 (Tipo)
    end if

vincula1 (tipo_punt (Tipo )):
    if Tipo != tipo_ident (__) then
        vincula1 (Tipo)
    end if

vincula2 (tipo_punt (Tipo )):
    if Tipo == tipo_ident (id) then
        Tipo.vinculo = vinculoDe (ts , id)
        if Tipo.vinculo != dec_type (__) then
            error
        end if
    else
        vincula2 (Tipo)
    end if

```

```

vincula1(tipo_bool()): noop

vincula2(tipo_bool()): noop

vincula1(tipo_int()): noop

vincula2(tipo_int()): noop

vincula1(tipo_real()): noop

vincula2(tipo_real()): noop

vincula1(tipo_string()): noop

vincula2(tipo_string()): noop

vincula1(tipo_ident(id)):
    $.vinculo = vinculoDe(ts,id)
    if $.vinculo != dec_type(_) then
        error
    end if

vincula2(tipo_ident(id)): noop

vincula(tipo_struct(LVar)):
    vincula1(LVar)

vincula(si_inst(LInst)):
    vincula(LInst):

vincula(no_inst()): noop

vincula(muchas_inst(LInst,Inst)):
    vincula(LInst)
    vincula(Inst)

vincula(una_inst(Inst)):
    vincula(Inst)

vincula1(si_pformal(LPFml)):
    vincula1(LPFml)

vincula2(si_pformal(LPFml)):
    vincula2(LPFml)

vincula1(no_pformal()): noop

vincula2(no_pformal()): noop

vincula1(muchos_pformal(LPFml,PFml)):
    vincula1(LPFml)
    vincula1(PFml)

vincula2(muchos_pformal(LPFml,PFml)):
    vincula2(LPFml)
    vincula2(PFml)

vincula1(un_pformal(PFml))
    vincula1(PFml)

```

```

vincula2(un_pformal(PFml))
    vincula2(PFml)

vincula1(pformal_ref(Tipo, id)):
    vincula1(Tipo)
    if contiene(ts, id) then
        error
    else
        inserta(ts, id, $)
    end if

vincula2(pformal_ref(Tipo, id)):
    vincula2(Tipo)

vincula1(pformal_noref(Tipo, id)):
    vincula1(Tipo)
    if contiene(ts, id) then
        error
    else
        inserta(ts, id, $)
    end if

vincula2(pformal_noref(Tipo, id)):
    vincula2(Tipo)

vincula(si_preales(LPReal)):
    vincula(LPReal)

vincula(no_preales()): noop

vincula(muchas_exp(LPReal, Exp)):
    vincula(LPReal)
    vincula(Exp)

vincula(una_exp(Exp)):
    vincula(Exp)

vincula(inst_eval(Exp)):
    vincula(Exp)

vincula(inst_if(Exp, Blo)):
    vincula(Exp)
    vincula(Blo)

vincula(inst_else(Exp, Blo1, Blo2)):
    vincula(Exp)
    vincula(Blo1)
    vincula(Blo2)

vincula(inst_while(Exp, Blo)):
    vincula(Exp)
    vincula(Blo)

vincula(inst_new(Exp)):
    vincula(Exp)

vincula(inst_delete(Exp)):
    vincula(Exp)

vincula(inst_read(Exp)):

```

```
vincula (Exp)

vincula (inst_write (Exp)) :
    vincula (Exp)

vincula (inst_call (id, PReales)) :
    $.vinculo = vinculoDe (ts, Id)
    if $.vinculo == false then
        error
    end if
    vincula (PReales)

vincula (inst_nl ()) : noop

vincula (inst_blo (Blo)) :
    vincula (Blo)

vincula (exp_asig (Opnd0, Opnd1)) :
    vincula (Opnd0)
    vincula (Opnd1)

vincula (exp_menor (Opnd0, Opnd1)) :
    vincula (Opnd0)
    vincula (Opnd1)

vincula (exp_menIgual (Opnd0, Opnd1)) :
    vincula (Opnd0)
    vincula (Opnd1)

vincula (exp_mayor (Opnd0, Opnd1)) :
    vincula (Opnd0)
    vincula (Opnd1)

vincula (exp_mayIgual (Opnd0, Opnd1)) :
    vincula (Opnd0)
    vincula (Opnd1)

vincula (exp_igual (Opnd0, Opnd1)) :
    vincula (Opnd0)
    vincula (Opnd1)

vincula (exp_dist (Opnd0, Opnd1)) :
    vincula (Opnd0)
    vincula (Opnd1)

vincula (exp_sum (Opnd0, Opnd1)) :
    vincula (Opnd0)
    vincula (Opnd1)

vincula (exp_resta (Opnd0, Opnd1)) :
    vincula (Opnd0)
    vincula (Opnd1)

vincula (exp_mult (Opnd0, Opnd1)) :
    vincula (Opnd0)
    vincula (Opnd1)

vincula (exp_div (Opnd0, Opnd1)) :
    vincula (Opnd0)
    vincula (Opnd1)
```

```
vincula (exp_mod (Opnd0, Opnd1)) :  
    vincula (Opnd0)  
    vincula (Opnd1)  
  
vincula (exp_and (Opnd0, Opnd1)) :  
    vincula (Opnd0)  
    vincula (Opnd1)  
  
vincula (exp_or (Opnd0, Opnd1)) :  
    vincula (Opnd0)  
    vincula (Opnd1)  
  
vincula (exp_menos (Exp)) :  
    vincula (Exp)  
  
vincula (exp_not (Exp)) :  
    vincula (Exp)  
  
vincula (inst_index (Opnd0, Opnd1)) :  
    vincula (Opnd0)  
    vincula (Opnd1)  
  
vincula (exp_reg (Exp, id)) :  
    vincula (Exp)  
  
vincula (exp_ind (Exp)) :  
    vincula (Exp)  
  
vincula (exp_true ()) : noop  
  
vincula (exp_false ()) : noop  
  
vincula (exp_litEnt (litEnt)) : noop  
  
vincula (exp_litReal (litReal)) : noop  
  
vincula (exp_litCad (litCad)) : noop  
  
vincula (exp_iden (id)) :  
    $.vinculo = vinculoDe (ts, Id)  
    if $.vinculo == false then  
        error  
    end if  
  
vincula (exp_null ()) : noop
```



## 2 | Especificación del procesamiento de comprobación de tipos

---

```
tipado(prog(Blo)):
    tipado(Blo)
    $.tipo = Blo.tipo

tipado(bloq(Decs, Insts)):
    tipado(Decs)
    tipado(Insts)
    $.tipo = ambos-ok(Decs.tipo, Insts.tipo)

tipado(no_decs()): $.tipo = ok

tipado(si_decs(LDecs)):
    tipado(LDecs)
    $.tipo = ok

tipado(muchas_decs(LDecs, Dec)):
    tipado(LDecs)
    tipado(Dec)
    $.tipo = ambos-ok(LDecs.tipo, Dec.tipo)

tipado(una_dec(Dec)):
    tipado(Dec)
    $.tipo = Dec.tipo

tipado(muchas_var(LVar, Var)):
    tipado(LVar)
    tipado(Var)
    $.tipo = ambos-ok(LVar.tipo, Var.tipo)

tipado(una_var(Var)):
    tipado(Var)
    $.tipo = Var.tipo

tipado(var(Tipo, id)):
    tipado(Tipo)
    $.tipo = Tipo.tipo

tipo(dec_simple(Var)):
    tipado(Var)
    $.tipo = ok

tipo(dec_type(Var)):
    tipado(Var)
    $.tipo = ok

tipo(dec_proc(id, PFmls, Blo)):
    tipado(PFmls)
    tipado(Blo)
    $.tipo = Blo.tipo

tipado(tipo_array(Tipo, litEnt)):
    if litEnt < 0 then
        $.tipo = error
    else
```

```
        tipado(Tipo)
        $.tipo = Tipo.tipo
    end if

tipado(tipo_punt(Tipo)):
    tipado(Tipo)
    $.tipo = Tipo.tipo

tipado(tipo_bool()): $.tipo = bool

tipado(tipo_int()): $.tipo = int

tipado(tipo_real()): $.tipo = real

tipado(tipo_string()): $.tipo = string

tipado(tipo_ident(id)):
    let $.vinculo = var(T,I) in
        $.tipo = T
    end let

tipado(tipo_struct(LVar)):
    tipado(LVar)
    $.tipo = LVar.tipo

tipado(si_inst(LInst)):
    tipado(LInst)
    $.tipo = ok

tipado(no_inst()): $.tipo = ok

tipado(muchas_inst(LInst, Inst)):
    tipado(LInst)
    tipado(Inst)
    $.tipo = ambos-ok(LInst.tipo, Inst.tipo)

tipado(una_inst(Inst)):
    tipado(Inst)
    $.tipo = Inst.tipo

tipado(si_pformal(LPFml)):
    tipado(LPFml)
    $.tipo = ok

tipado(no_pformal()): $.tipo = ok

tipado(muchos_pformal(LPFml, PFml)):
    tipado(LPFml)
    tipado(PFml)
    $.tipo = ambos-ok(LPFml.tipo, PFml.tipo)

tipado(un_pformal(PFml))
    tipado(PFml)
    $.tipo = PFml.tipo

tipado(si_preales(LPReal)):
    tipado(LPReal)
    $.tipo = ok

tipado(no_preales()): $.tipo = ok
```

```

tipado (muchas_exp (LPreal, Exp)):
    tipado (LPreal)
    tipado (Exp)
    $.tipo = ambos-ok (LPreal.tipo, Exp.tipo)

tipado (una_exp (Exp)):
    tipado (Exp)
    $.tipo = Exp.tipo

tipado (pformal_ref (Tipo, id)):
    tipado (Tipo)
    $.tipo = Tipo.tipo

tipado (pformal_noref (Tipo, id)):
    tipado (Tipo)
    $.tipo = Tipo.tipo

tipado (inst_eval (Exp)):
    tipado (Exp)
    $.tipo = ok

tipado (inst_if (Exp, Blo)):
    tipado (Exp)
    tipado (Blo)
    if Exp.tipo == bool ^ Bloq.tipo == ok then
        $.tipo = ok
    else
        $.tipo = error

tipado (inst_else (Exp, Blo1, Blo2)):
    tipado (Exp)
    tipado (Blo1)
    tipado (Blo2)
    if Exp.tipo == bool ^ Blo1 == ok ^ Blo2 == ok then
        $.tipo = ok
    else
        $.tipo = error
    end if

tipado (inst_while (Exp, Blo)):
    tipado (Exp)
    tipado (Blo)
    if Exp.tipo == bool ^ Bloq.tipo == ok then
        $.tipo = ok
    else
        $.tipo = error

tipado (inst_new (Exp)):
    tipado (Exp)
    if ref!(Exp.tipo) == tipo_punt(T) then
        $.tipo = ok
    else
        aviso-error (T)
        $.tipo = error
    end if

tipado (inst_delete (Exp)):
    tipado (Exp)
    if ref!(Exp.tipo) == tipo_punt(T) then

```

```

        $.tipo = ok
    else
        aviso-error(T)
        $.tipo = error
    end if

tipado(inst_read(Exp)):
    tipado(Exp)
    if es-designador(Exp) then
        if ref!(Exp.tipo) == literalEntero v ref!(Exp.tipo) == literalReal v ref!(Exp.tipo) == literalComplex then
            $.tipo = ok
        else
            $.tipo = error
        end if
    else
        $.tipo = error
    end if

tipado(inst_write(Exp)):
    tipado(Exp)
    if ref!(Exp.tipo) == literalEntero v ref!(Exp.tipo) == literalReal v ref!(Exp.tipo) == literalComplex then
        $.tipo = ok
    else
        $.tipo = error
    end if

tipado(inst_call(id, PReales)):
    if $.vinculo == dec_proc(id, PFmls, Blo) then
        if num_elems(PReales) != num_elems(PFmls)
            $.tipo = error
        else
            tipado_parametros(PReales, PFmls)
        end if
    else
        $.tipo = error
    end if
    tipado(PReales)
    $.tipo = ok

tipado_parametros(no_preales(), no_pformal()):
    return ok

tipado_parametros(si_preales(LPReal), si_pformal(LPFml)):
    return tipado_parametros(LPReal, LPFml)

tipado_parametros(muchas_exp(LPReal, Exp), muchas_pformal(LPFml, PFml)):
    return ambos_ok(tipado_parametros(LPReal, LPFml), tipado_parametros(Exp, PFml))

tipado_parametros(una_exp(Exp), pformal_ref(Tipo, id)):
    tipado(Exp)
    if compatibles(Exp.tipo, Tipo.tipo) entonces
        return ok
    else
        aviso-error(Tipo.tipo)
        return error
    end if

tipado_parametros(una_exp(Exp), pformal_noref(Tipo, id)):
    tipado(Exp)
    if es-designador(Exp) ^ compatibles(Exp.tipo, Tipo.tipo) entonces
        return ok
    end if

```

```

        else
            aviso-error (Tipo.tipo)
            return error

tipado(inst_nl()): $.tipo = nl

tipado(inst_blo(Blo)):
    tipado(Blo)
    $.tipo = Blo.tipo

tipado(exp_asig(Opnd0, Opnd1)):
    tipado(Opnd0)
    tipado(Opnd1)
    if es-designador(Opnd0) then
        if compatibles(Opnd0.tipo, Opnd1.tipo) then
            $.tipo = Opnd0.tipo
        else
            aviso-error(Opnd0.tipo, Opnd1.tipo)
            $.tipo = error
        end if
    else
        error
        $.tipo = error
    end if

tipado(exp_menor(Opnd0, Opnd1)):
    tipado-bin2(Opnd0, Opnd1, $)

tipado(exp_menIgual(Opnd0, Opnd1)):
    tipado-bin2(Opnd0, Opnd1, $)

tipado(exp_mayor(Opnd0, Opnd1)):
    tipado-bin2(Opnd0, Opnd1, $)

tipado(exp_mayIgual(Opnd0, Opnd1)):
    tipado-bin2(Opnd0, Opnd1, $)

tipado(exp_igual(Opnd0, Opnd1)):
    tipado-bin3(Opnd0, Opnd1, $)

tipado(exp_dist(Opnd0, Opnd1)):
    tipado-bin3(Opnd0, Opnd1, $)

tipado(exp_sum(Opnd0, Opnd1)):
    tipado-bin1(Opnd0, Opnd1, $)

tipado(exp_resta(Opnd0, Opnd1)):
    tipado-bin1(Opnd0, Opnd1, $)

tipado(exp_mult(Opnd0, Opnd1)):
    tipado-bin1(Opnd0, Opnd1, $)

tipado(exp_div(Opnd0, Opnd1)):
    tipado-bin1(Opnd0, Opnd1, $)

tipado(exp_mod(Opnd0, Opnd1)):
    tipado(Opnd0)
    tipado(Opnd1)
    if ref!(Opnd0.tipo) == literalEntero ^ ref!(Opnd1.tipo) == literalEntero then
        return literalEntero
    end if

```

```

    else
        return error
    end if

tipado(exp_and(Opnd0, Opnd1)):
    tipado(Opnd0)
    tipado(Opnd1)
    if ref!(Opnd0.tipo) == bool ^ ref!(Opnd1.tipo) == bool then
        return bool
    else
        return error
    end if

tipado(exp_or(Opnd0, Opnd1)):
    tipado(Opnd0)
    tipado(Opnd1)
    if ref!(Opnd0.tipo) == bool ^ ref!(Opnd1.tipo) == bool then
        return bool
    else
        return error
    end if

tipado(exp_menos(Exp)):
    tipado(Exp)
    if ref!(Exp.tipo) == literalEntero then
        return literalEntero
    if ref!(Exp.tipo) == literalReal then
        return literalReal
    else
        return error
    end if

tipado(exp_not(Exp)):
    tipado(Exp)
    if ref!(Exp.tipo) == bool then
        return bool
    else
        return error
    end if

tipado(inst_index(Opnd0, Opnd1)):
    tipado(Opnd0)
    tipado(Opnd1)
    if ref!(Opnd0.tipo) == tipo_array(T) ^ ref!(Opnd1.tipo) == literalEntero then
        return tipo_array(T)
    else
        return error
    end if

tipado(exp_reg(Exp, id)):
    tipado(Exp)
    let $.vinculo = var(T, I) in
        if ref!(Exp.tipo) == tipo_struct(T) then
            return T.tipo
        else
            return error
        end if
    end let

tipado(exp_ind(Exp)):
    tipado(Exp)
    if ref!(Exp.tipo) == tipo_punt(T) then
        $.tipo = ok
    end if

```

```

    else
        aviso-error(T)
        $.tipo = error
    end if

tipado(exp_true()): $.tipo = true

tipado(exp_false()): $.tipo = false

tipado(exp_litEnt(litEnt)): $.tipo = literalEntero

tipado(exp_litReal(litReal)): $.tipo = literalReal

tipado(exp_litCad(litCad)): $.tipo = literalCadena

tipado(exp_iden(id)):
    let $.vinculo = Dec in
        if Dec = var(Tipo,id) v Dec = pformal_ref(Tipo,id) v Dec = pformal_noref(Tipo,
            $.tipo = Tipo
        else
            aviso-error(id)
            $.tipo = error()
        end let

tipado(exp_null()): $.tipo = null

ambos-ok(T0,T1):
    if T0 == ok ^ T1 == ok then
        return ok
    else
        return error
    end if

aviso-error(T0,T1):
    if T0 != error ^ T1 != error then
        error
    end if

aviso-error(T):
    if T != error then
        error
    end if

ref!(T):
    if T == Ref(I) then
        let T.vinculo = Dec_type(Var(T',I)) in
            return ref!(T')
        end let
    else
        return T
    end if

tipado-bin1(E0,E1,E):
    tipado(E0)
    tipado(E1)
    E.tipo = tipo-bin1(E0.tipo,E1.tipo)

tipo-bin1(T0,T1):
    if compatibles(T0,T1) then
        return T0

```

```

    else if ref!(T1) == literalReal then
        if ref!(T0) == literalReal v ref!(T0) == literalEntero then
            return literalReal
        else
            return error
        end if
    else if ref!(T0) == literalReal then
        if ref!(T1) == literalReal v ref!(T1) == literalEntero then
            return literalReal
        else
            return error
        end if
    else
        return error
    end if

tipado-bin2(E0,E1,E):
    tipado(E0)
    tipado(E1)
    E.tipo = tipo-bin2(E0.tipo,E1.tipo)

tipo-bin2(T0,T1):
    if (ref!(T0) == literalReal v ref!(T0) == literalEntero) ^ (ref!(T1) == literalReal v ref!(T1) == literalEntero) then
        return bool
    else if ref!(T0) == bool ^ ref!(T1) == bool then
        return bool
    else if ref!(T0) == literalCadena ^ ref!(T1) == literalCadena then
        return bool
    else
        return error
    end if

tipado-bin3(E0,E1,E):
    tipado(E0)
    tipado(E1)
    E.tipo = tipo-bin3(E0.tipo,E1.tipo)

tipo-bin3(T0,T1):
    if (ref!(T0) == literalReal v ref!(T0) == literalEntero) ^ (ref!(T1) == literalReal v ref!(T1) == literalEntero) then
        return bool
    else if ref!(T0) == bool ^ ref!(T1) == bool then
        return bool
    else if ref!(T0) == literalCadena ^ ref!(T1) == literalCadena then
        return bool
    else if (ref!(T0) == tipo_punt(T) v ref!(T0) == null) ^ (ref!(T1) == tipo_punt(T) v ref!(T1) == null) then
        return bool
    else
        return error
    end if

compatibles(T1,T2):
    let T1' = ref!(T1) ^ T2' = ref!(T2) in
        if T1' == T2' then
            return true;
        else if T1 == literalReal ^ (T2 == literalEntero v T2 == literalReal) then
            return true;
        else if T1 == tipo_array(T,literalEntero) ^ T2 == tipo_array(T,literalEntero) then
            return true;
        else if T1 == tipo_struct(LVar) ^ T2 == tipo_struct(LVar) then
            return true;
        else
            return false;
        end if
    end let

```



```
    else if T1== tipo_punt(T) ^ T2== null then
        return true;
    else if T1== tipo_punt(T) ^ T2 == tipo_punt(T) then
        return true;
    else
        return false
    end if
end let

es--designador(E):
    return E = id(v) v E = elem1(E') v E = elem2(E')
```

### 3 | Especificación del procesamiento de asignación de espacio

---

```
global dir = 0 //contador de direcciones
global nivel = 0 //nivel de anidamiento
```

```
asig-espacio(prog(Blo)):
    asig-espacio(Blo)
```

```
asig-espacio(bloq(Decs, Insts)):
    dir_ant = dir
    asig-espacio(Decs)
    asig-espacio(Insts)
    dir = dir_ant
```

```
asig-espacio(si_decs(LDecs)):
    asig-espacio1(Ldecs)
    asig-espacio2(Ldecs)
```

```
asig-espacio(no_decs()): skip
```

```
asig-espacio1(muchas_decs(LDecs, Dec)):
    asig-espacio1(LDecs)
    asig-espacio1(Dec)
```

```
asig-espacio2(muchas_decs(LDecs, Dec)):
    asig-espacio2(LDecs)
    asig-espacio2(Dec)
```

```
asig-espacio1(una_dec(Dec)):
    asig-espacio1(Dec)
```

```
asig-espacio2(una_dec(Dec)):
    asig-espacio2(Dec)
```

```
asig-espacio1(muchas_var(LVar, Var)):
    asig-espacio1(LVar)
    asig-espacio1(Var)
```

```
asig-espacio2(muchas_var(LVar, Var)):
    asig-espacio2(LVar)
    asig-espacio2(Var)
```

```
asig-espacio1(una_var(Var)):
    asig-espacio1(Var)
```

```
asig-espacio2(una_var(Var)):
    asig-espacio2(Var)
```

```
asig-espacio1(var(Tipo, id)):
    $.dir = dir
    $.nivel = nivel
    asig-espacio-tipo(Tipo)
    dir += Tipo.tam
```

```
asig-espacio2(var(Tipo, id)): skip
```

```

asig-espacio1(dec_simple(Var)):
    asig-espacio1(Var)

asig-espacio2(dec_simple(Var)):
    asig-espacio2(Var)

asig-espacio1(dec_type(Var)):
    asig-espacio1(Var)

asig-espacio2(dec_type(Var)):
    asig-espacio2(Var)

asig-espacio1(dec_proc(id,PFmls,Blo)):
    ant_dir = dir
    nivel = nivel + 1
    $.nivel = nivel
    dir = 0
    asigna_espacio1(PFmls)
    asigna_espacio1(Blo)
    $.tam_datos = dir
    dir = ant_dir
    nivel = nivel - 1

asig-espacio2(dec_proc(id,PFmls,Blo)):
    asigna_espacio2(PFmls)
    asigna_espacio2(Blo)

asig-espacio-tipo(Tipo):
    si indefinido(Tipo.tam)
        asig-espacio-tipo1(Tipo)
        asig-espacio-tipo2(Tipo)
    fin s

asig-espacio-tipo1(tipo_array(Tipo,litEnt)):
    $.tam = litEnt.tam
    si Tipo != tipo_ident(_)
        asig-espacio-tipo1(Tipo)
    fin si

asig-espacio-tipo2(tipo_array(Tipo,litEnt)):
    si Tipo = tipo_ident(id)
        Tipo.vinculo = vinculoDe(ts,id)
        sea Tipo.vinculo = dec_type(Tipo,_)
        asig-espacio-tipo(Tipo)
        $.tam= Tipo.tam*litEnt.tam
    fin si

asig-espacio-tipo1(tipo_punt(Tipo)):
    $.tam = 1
    si Tipo != tipo_ident(_)
        asig-espacio-tipo1(Tipo)
    fin si

asig-espacio-tipo2(tipo_punt(Tipo)):
    si Tipo = tipo_ident(id)
        sea Tipo.vinculo = dec_type(Tipo,_)

```

```

        asig-espacio-tipo( Tipo )
        $.tam= Tipo .tam
    fin si

asig-espacio-tipo1(tipo_bool()):
    $.tam = 1

asig-espacio-tipo2(tipo_bool()): skip

asig-espacio-tipo1(tipo_int()):
    $.tam = 1

asig-espacio-tipo2(tipo_int()): skip

asig-espacio-tipo1(tipo_real()):
    $.tam = 1

asig-espacio-tipo2(tipo_real()): skip

asig-espacio-tipo1(tipo_string()):
    $.tam = 1

asig-espacio-tipo2(tipo_string()): skip

asig-espacio-tipo1(tipo_ident(id)):
    asig-espacio-tipo1($.vinculo)
    sea $.vinculo = dec_type(Tipo,_) en
        $.tam = Tipo.tam

asig-espacio-tipo2(tipo_ident(id)): skip

asig-espacio(tipo_struct(LVar)):
    asig-espacio(LVar)

asig-espacio(si_inst(LInst)):
    asig-espacio(LInst):

asig-espacio(no_inst()): skip

asig-espacio(muchas_inst(LInst, Inst)):
    asig-espacio(LInst)
    asig-espacio(Inst)

asig-espacio(una_inst(Inst)):
    asig-espacio(Inst)

asig-espacio1(si_pformal(LPFml)):
    asig-espacio1(LPFml)

asig-espacio2(si_pformal(LPFml)):
    asig-espacio2(LPFml)

asig-espacio1(no_pformal()): skip

asig-espacio2(no_pformal()): skip

asig-espacio1(muchos_pformal(LPFml, PFml)):
    asig-espacio1(LPFml)
    asig-espacio1(PFml)

```

```

asig-espacio2 (muchos_pformal (LPFml, PFml)) :
    asig-espacio2 (LPFml)
    asig-espacio2 (PFml)

asig-espacio1 (un_pformal (PFml))
    asig-espacio1 (PFml)

asig-espacio2 (un_pformal (PFml))
    asig-espacio2 (PFml)

asig-espacio1 (pformal_ref (Tipo, id)) :
    $.dir = dir
    $.nivel = nivel
    asig-espacio-tipo (Tipo)
    dir += Tipo.tam

asig-espacio2 (pformal_ref (Tipo, id)) :
    asig-espacio-tipo (Tipo)

asig-espacio1 (pformal_noref (Tipo, id)) :
    $.dir = dir
    $.nivel = nivel
    asig-espacio-tipo (Tipo)
    dir += Tipo.tam

asig-espacio2 (pformal_noref (Tipo, id)) :
    asig-espacio-tipo (Tipo)

asig-espacio (si_preales (LPReal)) :
    asig-espacio (LPReal)

asig-espacio (no_preales ()) : skip

asig-espacio (muchas_exp (LPReal, Exp)) :
    asig-espacio (LPReal)
    asig-espacio (Exp)

asig-espacio (una_exp (Exp)) :
    asig-espacio (Exp)

asig-espacio (inst_eval (Exp)) : skip

asig-espacio (inst_if (Exp, Blo)) :
    asig-espacio (Blo)

asig-espacio (inst_else (Exp, Blo1, Blo2)) :
    asig-espacio (Blo1)
    asig-espacio (Blo2)

asig-espacio (inst_while (Exp, Blo)) :
    asig-espacio (Blo)

asig-espacio (inst_new (Exp)) : skip

asig-espacio (inst_delete (Exp)) : skip

asig-espacio (inst_read (Exp)) : skip

```

```
asig-espacio(inst_write(Exp)): skip

asig-espacio(inst_call(id, PReales)): skip

asig-espacio(inst_nl()): skip

asig-espacio(inst_blo(Blo)):
    asig-espacio(Blo)

asig-espacio(exp_asig(Opnd0, Opnd1)):
    asig-espacio(Opnd0)
    asig-espacio(Opnd1)

asig-espacio(exp_menor(Opnd0, Opnd1)):
    asig-espacio(Opnd0)
    asig-espacio(Opnd1)

asig-espacio(exp_menIgual(Opnd0, Opnd1)):
    asig-espacio(Opnd0)
    asig-espacio(Opnd1)

asig-espacio(exp_mayor(Opnd0, Opnd1)):
    asig-espacio(Opnd0)
    asig-espacio(Opnd1)

asig-espacio(exp_mayIgual(Opnd0, Opnd1)):
    asig-espacio(Opnd0)
    asig-espacio(Opnd1)

asig-espacio(exp_igual(Opnd0, Opnd1)):
    asig-espacio(Opnd0)
    asig-espacio(Opnd1)

asig-espacio(exp_dist(Opnd0, Opnd1)):
    asig-espacio(Opnd0)
    asig-espacio(Opnd1)

asig-espacio(exp_sum(Opnd0, Opnd1)):
    asig-espacio(Opnd0)
    asig-espacio(Opnd1)

asig-espacio(exp_resta(Opnd0, Opnd1)):
    asig-espacio(Opnd0)
    asig-espacio(Opnd1)

asig-espacio(exp_mult(Opnd0, Opnd1)):
    asig-espacio(Opnd0)
    asig-espacio(Opnd1)

asig-espacio(exp_div(Opnd0, Opnd1)):
    asig-espacio(Opnd0)
    asig-espacio(Opnd1)

asig-espacio(exp_mod(Opnd0, Opnd1)):
    asig-espacio(Opnd0)
    asig-espacio(Opnd1)

asig-espacio(exp_and(Opnd0, Opnd1)):
    asig-espacio(Opnd0)
    asig-espacio(Opnd1)
```

```
asig-espacio (exp_or (Opnd0, Opnd1)) :  
    asig-espacio (Opnd0)  
    asig-espacio (Opnd1)  
  
asig-espacio (exp_menos (Exp)) :  
    asig-espacio (Exp)  
  
asig-espacio (exp_not (Exp)) :  
    asig-espacio (Exp)  
  
asig-espacio (inst_index (Opnd0, Opnd1)) :  
    asig-espacio (Opnd0)  
    asig-espacio (Opnd1)  
  
asig-espacio (exp_reg (Exp, id)) :  
    asig-espacio ($.vinculo)  
    sea $.vinculo = tipo_struct (Tipo, _) en  
        $.tam = Tipo.tam  
  
asig-espacio (exp_ind (Exp)) :  
    asig-espacio (Exp)  
  
asig-espacio (exp_true ()) : skip  
  
asig-espacio (exp_false ()) : skip  
  
asig-espacio (exp_litEnt (litEnt)) : skip  
  
asig-espacio (exp_litReal (litReal)) : skip  
  
asig-espacio (exp_litCad (litCad)) : skip  
  
asig-espacio (exp_iden (id)) :  
    asig-espacio ($.vinculo)  
    sea $.vinculo = dec_type (Tipo, _) en  
        $.tam = Tipo.tam  
  
asig-espacio (exp_null ()) : skip
```

## 4 | Instrucciones de la máquina-p necesarias para soportar la traducción de Tiny a código-p

### 4.0.1. Instrucciones Aritmético-Lógicas

Estas instrucciones:

- *Desapilan los argumentos de la pila de evaluación (los argumentos aparecen en la pila en orden inverso; por ejemplo, si la operación necesita dos argumentos, en la cima estará el 2º argumento, en la sub-cima el 1er argumento).*
- *Realizan la operación.*
- *Apilan el resultado en la pila de evaluación.*

Cuadro 4.0.1: Instrucciones Aritmético-Lógicas

suma	Desapila los 2 primeros argumentos de la pila y apila su suma
resta	Desapila los 2 primeros argumentos de la pila y apila su resta
menos	Desapila el primer argumento de la pila y apila su negativo
mult	Desapila los 2 primeros argumentos de la pila y apila su multiplicación
div	Desapila los 2 primeros argumentos de la pila y apila su división
mod	Desapila los 2 primeros argumentos de la pila y apila su módulo
and	Desapila los 2 primeros argumentos de la pila y apila el resultado de la operación and
or	Desapila los 2 primeros argumentos de la pila y apila el resultado de la operación or
not	Desapila el primer argumento de la pila y apila su contrario
asig	Desapila los 2 primeros argumentos de la pila y apila el resultado de la operación asignación
menor	Desapila los 2 primeros argumentos de la pila y apila true si el primer argumento es menor
menIgual	Desapila los 2 primeros argumentos de la pila y apila true si el primer argumento es menor o igual
mayor	Desapila los 2 primeros argumentos de la pila y apila true si el primer argumento es mayor
mayIgual	Desapila los 2 primeros argumentos de la pila y apila true si el primer argumento es mayor o igual

Continúa en la siguiente página



Cuadro 4.0.1: Instrucciones Aritmético-Lógicas (Continuación)

igual	Desapila los 2 primeros argumentos de la pila y apila true si ambos argumentos son iguales
distinto	Desapila los 2 primeros argumentos de la pila y apila true si los argumentos son distintos

#### 4.0.2. Instrucciones de movimiento de datos

Cuadro 4.0.2: Instrucciones de movimiento de datos

apilaInt(v)	Apila el valor entero v en la pila de evaluación
apilaIden(v)	Apila el valor del identificador v en la pila de evaluación
apilaBool(v)	Apila el valor booleano v en la pila de evaluación
apilaReal(v)	Apila el valor real v en la pila de evaluación
apilaString(v)	Apila la cadena de caracteres v en la pila de evaluación
desapila(v)	desapila el valor de la cima de la pila de evaluación
apilaInd	Desapila una dirección (dir) de la pila de evaluación, y apila (en dicha pila) el contenido de la celda (dir) en la memoria de datos
desapilaInd	Desapila el valor v y una dirección (dir) de la pila de evaluación (primero v, después d), y actualiza el contenido de la celda dir en la memoria de datos con el valor de v
mueve(n)	Desapila dos direcciones dir1 y dir0 de la pila de evaluación en ese orden y copia el contenido de las n celdas consecutivas desde la dirección dir1 a las correspondientes n celdas que comienzan en la dirección dir0.
int2real()	desapila un 'entero' y apila el valor 'real' equivalente.

#### 4.0.3. Instrucciones de salto

Cuadro 4.0.3: Instrucciones de salto

irA(d)	Salto incondicional a la dirección d.
irF(d)	Desapila el valor v. Si es falso salta a la dirección d.
irV(d)	Desapila el valor v. Si es verdadero salta a la dirección d.
irInd	Desapila una dirección d de la pila de evaluación, y realiza un salto incondicional a dicha dirección

#### 4.0.4. Instrucciones de gestión de memoria dinámica

Cuadro 4.0.4: Instrucciones de gestión de memoria dinámica

<code>alloc(t)</code>	e reserva una zona de memoria adecuada para almacenar valores del tipo <code>t</code> . La operación en sí devuelve la dirección de comienzo de dicha zona de memoria.
<code>dealloc(d,t)</code>	Se notifica que la zona de memoria que comienza en <code>d</code> y que permite almacenar valores del tipo <code>t</code> queda liberada
<code>fetch(d)</code>	Devuelve el valor almacenado en la celda direccionada por <code>d</code> .
<code>store(d,v)</code>	Almacena <code>v</code> en la celda direccionada por <code>d</code> .
<code>copy(d,d',t)</code>	Copia el valor del tipo <code>t</code> que se encuentra almacenado a partir de la dirección <code>d'</code> en el bloque que comienza a partir de la dirección <code>d</code>
<code>indx(d,i,t)</code>	Considera que, a partir de <code>d</code> , comienza un array cuyos elementos son valores del tipo <code>t</code> , y devuelve la dirección de comienzo del elemento <code>i</code> -ésimo de dicho array.
<code>acc(d,c,t)</code>	Con <code>t</code> un tipo record. Considera que, a partir de <code>d</code> , está almacenado un registro de tipo <code>t</code> , que contiene un campo <code>c</code> . Devuelve la dirección de comienzo de dicho campo.
<code>dir(u)</code>	Dirección de la variable <code>u</code> .

#### 4.0.5. Instrucciones de Soporte de la ejecución de procedimientos

Cuadro 4.0.5: Instrucciones de Soporte de la ejecución de procedimientos

<code>activa(n,t,dir)</code>	Reserva espacio en el segmento de pila de registros de activación para ejecutar un procedimiento que tiene nivel de anidamiento <code>n</code> y tamaño de datos locales <code>t</code> . Así mismo, almacena en la zona de control de dicho registro <code>dir</code> como dirección de retorno. También almacena en dicha zona de control el valor del display de nivel <code>n</code> . Por último, apila en la pila de evaluación la dirección de comienzo de los datos en el registro creado
<code>apilad(n)</code>	Apila en la pila de evaluación el valor del display de nivel <code>n</code>
<code>desapilad(n)</code>	Desapila una dirección <code>dir</code> de la pila de evaluación en el display de nivel <code>n</code>
<code>desactiva(n,t)</code>	Libera el espacio ocupado por el registro de activación actual, restaurando adecuadamente el estado de la máquina. <code>n</code> indica el nivel de anidamiento del procedimiento asociado; <code>t</code> el tamaño de los datos locales. De esta forma, la instrucción: (i) apila en la pila de evaluación la dirección de retorno; (ii) restaura el valor del display de nivel <code>n</code> al antiguo valor guardado en el registro; (iii) decrementa el puntero de pila de registros de activación en el tamaño ocupado por el registro

Continúa en la siguiente página

Cuadro 4.0.5: Instrucciones de Soporte de la ejecución de procedimientos (Continuación)

dup	Consulta el valor $v$ de la cima de la pila y apila de nuevo dicho valor.
stop	Detiene la máquina-p

## 5 | Especificación del procesamiento de etiquetado

---

```

var etq = 0

etiquetado(prog(Blo)):
    etiquetado(Blo)

etiquetado(bloq(Decs, Insts)):
    $.prim = etq
    etiquetado(Insts)
    etq++
    while != es-vacia(sub_pendientes)
        sub = cima(sub_pendientes)
        desapila(sub_pendientes)
        let sub = dec_proc(id, Param, Decs, Is) in
            sub.prim = etq
            etq++
            recolecta_subs(Decs)
            etiquetado(Is)
            etq+=2
            sub.sig = etq
        end let
    end while
    $.sig = etq

etiquetado(si_decs(LDecs)):
    $.prim = etq
    etiquetado(Ldecs)
    $.sig = etq

etiquetado(no_decs()): noop

etiquetado(muchas_decs(LDecs, Dec)):
    $.prim = etq
    etiquetado(LDecs)
    etiquetado(Dec)
    $.sig = etq

etiquetado(una_dec(Dec)):
    $.prim = etq
    etiquetado(Dec)
    $.sig = etq

etiquetado(muchas_var(LVar, Var)):
    $.prim = etq
    etiquetado(LVar)
    etiquetado(Var)
    $.sig = etq

etiquetado(una_var(Var)):
    $.prim = etq
    etiquetado(Var)
    $.sig = etq

etiquetado(var(Tipo, id)):

```

```
$.prim = etq
etiquetado(Tipo)
$.sig = etq

etiquetado(dec_simple(Var)):
$.prim = etq
etiquetado(Var)
$.sig = etq

etiquetado(dec_type(Var)):
$.prim = etq
etiquetado(Var)
$.sig = etq

etiquetado(dec_proc(id,PFmls,Blo)):
$.prim = etq
etiquetado(PFmls)
etiquetado(Blo)
$.sig = etq

etiquetado(tipo_array(Tipo,litEnt)):
$.prim = etq
etiquetado(Tipo)
$.sig = etq

etiquetado(tipo_punt(Tipo)):
$.prim = etq
etiquetado(Tipo)
$.sig = etq

etiquetado(tipo_bool()): noop

etiquetado(tipo_int()): noop

etiquetado(tipo_real()): noop

etiquetado(tipo_string()): noop

etiquetado(tipo_ident(id)):
$.prim = etq
etq++
$.sig = etq

etiquetado(tipo_struct(LVar)):
$.prim = etq
etiquetado(LVar)
$.sig = etq

etiquetado(si_inst(LInst)):
$.prim = etq
etiquetado(LInst):
$.sig = etq

etiquetado(no_inst()): noop

etiquetado(muchas_inst(LInst,Inst)):
$.prim = etq
etiquetado(LInst)
etiquetado(Inst)
$.sig = etq
```

```

etiquetado (una_inst (Inst)) :
    $.prim = etq
    etiquetado (Inst)
    $.sig = etq

etiquetado (si_pformal (LPFml)) :
    $.prim = etq
    etiquetado (LPFml)
    $.sig = etq

etiquetado (no_pformal ()) : noop

etiquetado (muchos_pformal (LPFml, PFml)) :
    $.prim = etq
    etiquetado (LPFml)
    etiquetado (PFml)
    $.sig = etq

etiquetado (un_pformal (PFml))
    $.prim = etq
    etiquetado (PFml)
    $.sig = etq

etiquetado (pformal_ref (Tipo, id)) :
    $.prim = etq
    etiquetado (Tipo)
    $.sig = etq

etiquetado (pformal_noref (Tipo, id)) :
    $.prim = etq
    etiquetado (Tipo)
    $.sig = etq

etiquetado (si_preales (LPReal)) :
    $.prim = etq
    etiquetado (LPReal)
    $.sig = etq

etiquetado (no_preales ()) : noop

etiquetado (muchas_exp (LPReal, Exp)) :
    $.prim = etq
    etiquetado (LPReal)
    etq++
    etiquetado (Exp)
    etiquetado_acc_val (Exp)
    etq++
    $.sig = etq

etiquetado (una_exp (Exp)) :
    $.prim = etq
    etiquetado (Exp)
    etq += 2
    $.sig = etq

etiquetado (inst_eval (Exp)) :
    $.prim = etq
    etiquetado (Exp)

```

```
    etq +=2
    $.sig = etq

etiquetado ( inst_if (Exp, Blo) ):
    $.prim = etq
    etiquetado (Exp)
    etiquetado-acc-val (Exp)
    etq++
    etiquetado (Blo)
    etq++
    $.sig = etq

etiquetado ( inst_else (Exp, Blo1, Blo2) ):
    $.prim = etq
    etiquetado (Exp)
    etiquetado-acc-val (Exp)
    etq++
    etiquetado (Blo1)
    etq++
    etiquetado (Blo2)
    etq++
    $.sig = etq

etiquetado ( inst_while (Exp, Blo) ):
    $.prim = etq
    etiquetado (Exp)
    etiquetado-acc-val (Exp)
    etq++
    etiquetado (Blo)
    etq++
    $.sig = etq

etiquetado ( inst_new (Exp) ):
    $.prim = etq
    etiquetado (Exp)
    etq +=2
    $.sig = etq

etiquetado ( inst_delete (Exp) ):
    $.prim = etq
    etiquetado (Exp)
    etq +=2
    $.sig = etq

etiquetado ( inst_read (Exp) ):
    $.prim = etq
    etiquetado (Exp)
    etq +=2
    $.sig = etq

etiquetado ( inst_write (Exp) ):
    $.prim = etq
    etiquetado (Exp)
    etq +=2
    $.sig = etq

etiquetado ( inst_call (id, PReales) ):
    $.prim = etq
    etq++
    etiquetado-paso-param ($.vinculo, PReales)
```

```

    etq++
    $.sig = etq

etiquetado(inst_nl()): noop

etiquetado(inst_blo(Blo)):
    $.prim = etq
    etiquetado(Blo)
    $.sig = etq

etiquetado(exp_asig(Opnd0, Opnd1)):
    $.prim = etq
    etiquetado-opnds(Opnd0, Opnd1)
    etq++
    $.sig = etq

etiquetado(exp_menor(Opnd0, Opnd1)):
    $.prim = etq
    etiquetado-opnds(Opnd0, Opnd1)
    etq++
    $.sig = etq

etiquetado(exp_menIgual(Opnd0, Opnd1)):
    $.prim = etq
    etiquetado-opnds(Opnd0, Opnd1)
    etq++
    $.sig = etq

etiquetado(exp_mayor(Opnd0, Opnd1)):
    $.prim = etq
    etiquetado-opnds(Opnd0, Opnd1)
    etq++
    $.sig = etq

etiquetado(exp_mayIgual(Opnd0, Opnd1)):
    $.prim = etq
    etiquetado-opnds(Opnd0, Opnd1)
    etq++
    $.sig = etq

etiquetado(exp_igual(Opnd0, Opnd1)):
    $.prim = etq
    etiquetado-opnds(Opnd0, Opnd1)
    etq++
    $.sig = etq

etiquetado(exp_dist(Opnd0, Opnd1)):
    $.prim = etq
    etiquetado-opnds(Opnd0, Opnd1)
    etq++
    $.sig = etq

etiquetado(exp_sum(Opnd0, Opnd1)):
    $.prim = etq
    etiquetado-opnds(Opnd0, Opnd1)
    etq++
    $.sig = etq

etiquetado(exp_resta(Opnd0, Opnd1)):
    $.prim = etq

```



```

    etiquetado-opnds (Opnd0, Opnd1)
    etq++
    $.sig = etq

etiquetado (exp_mult (Opnd0, Opnd1)) :
    $.prim = etq
    etiquetado-opnds (Opnd0, Opnd1)
    etq++
    $.sig = etq

etiquetado (exp_div (Opnd0, Opnd1)) :
    $.prim = etq
    etiquetado-opnds (Opnd0, Opnd1)
    etq++
    $.sig = etq

etiquetado (exp_mod (Opnd0, Opnd1)) :
    $.prim = etq
    etiquetado-opnds (Opnd0, Opnd1)
    etq++
    $.sig = etq

etiquetado (exp_and (Opnd0, Opnd1)) :
    $.prim = etq
    etiquetado-opnds (Opnd0, Opnd1)
    etq++
    $.sig = etq

etiquetado (exp_or (Opnd0, Opnd1)) :
    $.prim = etq
    etiquetado-opnds (Opnd0, Opnd1)
    etq++
    $.sig = etq

etiquetado (exp_menos (Exp)) :
    $.prim = etq
    etiquetado (Exp)
    etiquetado-acc-val (Exp)
    $.sig = etq

etiquetado (exp_not (Exp)) :
    $.prim = etq
    etiquetado (Exp)
    etiquetado-acc-val (Exp)
    etq += 2
    $.sig = etq

etiquetado (inst_index (Opnd0, Opnd1)) :
    $.prim = etq
    etiquetado-opnds (Opnd0, Opnd1)
    etq++
    $.sig = etq

etiquetado (exp_reg (Exp, id)) :
    $.prim = etq
    etq++
    etiquetado (Exp)
    etq++
    $.sig = etq

```

```

etiquetado(exp_ind(Exp)):
    $.prim = etq
    etiquetado(Exp)
    etq++
    $.sig = etq

etiquetado(exp_true()): noop

etiquetado(exp_false()): noop

etiquetado(exp_litEnt(litEnt)):
    $.prim = etq
    etq++
    $.sig = etq

etiquetado(exp_litReal(litReal)):
    $.prim = etq
    etq++
    $.sig = etq

etiquetado(exp_litCad(litCad)):
    $.prim = etq
    etq++
    $.sig = etq

etiquetado(exp_iden(id)):
    $.prim = etq
    etiquetado-acc-var(id.vinculo)
    $.sig = etq

etiquetado(exp_null()): noop

etiquetado-cod-opns(Opnd0, Opnd1):
    etiquetado(Opnd0)
    etiquetado-acc-val(Opnd0)
    etiquetado(Opnd1)
    etiquetado-acc-val(Opnd1)

etiquetado-acc-id(dec_var(Tipo, id)):
    if $.nivel = 0 then
        etq++
    else
        etiquetado-acc-var($)
    end if

etiquetado-acc-id(pval(Tipo, id)):
    etiquetado-acc-var($)

etiquetado-acc-id(pref(Tipo, id)):
    etiquetado-acc-var($)
    etq++

etiquetado-acc-var(V):
    etq += 3

```

```
etiquetado-acc-val(Exp):  
  if es-designador(Exp) then  
    etq++  
  end if  
  
etiquetado-paso-param(dec_proc(id,Param,Decs,Is),PReales):  
  etq += 2  
  etiquetado(PReales)  
  etq++
```

## 6 | Especificación del procesamiento de generación de código

---

```

var sub_pendientes = pila_vacia()

gen_cod(prog(Blo)):
  gen_cod(Blo)

gen_cod(bloq(Decs, Insts)):
  recolecta_procs(Decs)
  gen_cod(Insts)
  emit_stop()
  while not es_vacia(sub_pendientes)
    sub = cima(sub_pendientes)
    desapila(sub_pendientes)
    let sub = dec_proc(id, PFml, Decs, Is) in
      emit_desapilad(sub.nivel)
      recolecta_subs(Decs)
      gen_cod(Is)
      emit_desactiva(sub.nivel, sub.tam)
      emit_ir_ind()
    end let
  end while

gen_cod(si_inst(LInst)):
  gen_cod(LInst)

gen_cod(no_inst()): noop

gen_cod(muchas_inst(LInst, Inst)):
  gen_cod(LInst)
  gen_cod(Inst)

gen_cod(una_inst(Inst)):
  gen_cod(Inst)

gen_cod(si_preales(LPReal)):
  gen_cod(LPReal)

gen_cod(no_preales()): noop

gen_cod(muchas_exp(LPReal, Exp)):
  gen_cod(LPReal)
  gen_cod(Exp)

gen_cod(una_exp(Exp)):
  gen_cod(Exp)

gen_cod(inst_eval(Exp)):
  gen_cod(Exp)
  gen_acc_val(Exp)

gen_cod(inst_if(Exp, Blo)):
  gen_cod(Exp)
  gen_acc_val(Exp)
  emit_ir_f($.sig)

```

```

    gen_cod(Blo)

gen_cod(inst_else(Exp, Blo1, Blo2)):
    gen_cod(Exp)
    gen_acc_val(Exp)
    emit_ir_v($.sig)
    gen_cod(Blo1)
    emit_ir_f($.sig)
    gen_cod(Blo2)

gen_cod(inst_while(Exp, Blo)):
    gen_cod(Exp)
    gen_acc_val(Exp)
    emit_ir_f($.sig)
    gen_cod(Blo)
    emit_ir_a($.prim)

gen_cod(inst_new(Exp)):
    gen_cod(Exp)
    let ref!(Exp.tipo) = tipo_punt(T) in
        emit_alloc(T.tam)
    end
    emit_desapila_ind()

gen_cod(inst_delete(Exp)):
    gen_cod(Exp)
    emit_apila_ind()
    let ref!(Exp.tipo) = tipo_punt(T) in
        emit_dealloc(T.tam)
    end

gen_cod(inst_read(Exp)):
    gen_cod(Exp)

gen_cod(inst_write(Exp)):
    gen_cod(Exp)

gen_cod(inst_call(id, PReales)):
    emit_activa($.vinculo.nivel, $.vinculo.tam, $.sig)
    gen_paso_PFml($.vinculo, E)
    emit_ir_a($.vinculo.prim)

gen_cod(inst_nl()):
    emit_nl

gen_cod(inst_blo(Blo)):
    gen_cod(Blo)

gen_cod(exp_asig(Opnd0, Opnd1)):
    gen_cod(Opnd0)
    gen_cod(Opnd1)
    if es_designador(E) then
        emit_copia(E.tipo.tam)
    else
        emit_desapila_ind()
    end

gen_cod(exp_menor(Opnd0, Opnd1)):
    gen_cod(Opnd0)
    gen_acc_val(Opnd0)

```

```
    gen_cod(Opnd1)
    gen_acc_val(Opnd1)
    emit menor

gen_cod(exp_menIgual(Opnd0, Opnd1)) :
    gen_cod(Opnd0)
    gen_acc_val(Opnd0)
    gen_cod(Opnd1)
    gen_acc_val(Opnd1)
    emit menIgual

gen_cod(exp_mayor(Opnd0, Opnd1)) :
    gen_cod(Opnd0)
    gen_acc_val(Opnd0)
    gen_cod(Opnd1)
    gen_acc_val(Opnd1)
    emit mayor

gen_cod(exp_mayIgual(Opnd0, Opnd1)) :
    gen_cod(Opnd0)
    gen_acc_val(Opnd0)
    gen_cod(Opnd1)
    gen_acc_val(Opnd1)
    emit mayIgual

gen_cod(exp_igual(Opnd0, Opnd1)) :
    gen_cod(Opnd0)
    gen_acc_val(Opnd0)
    gen_cod(Opnd1)
    gen_acc_val(Opnd1)
    emit igual

gen_cod(exp_dist(Opnd0, Opnd1)) :
    gen_cod(Opnd0)
    gen_acc_val(Opnd0)
    gen_cod(Opnd1)
    gen_acc_val(Opnd1)
    emit distinto

gen_cod(exp_sum(Opnd0, Opnd1)) :
    gen_cod(Opnd0)
    gen_acc_val(Opnd0)
    gen_cod(Opnd1)
    gen_acc_val(Opnd1)
    emit suma

gen_cod(exp_resta(Opnd0, Opnd1)) :
    gen_cod(Opnd0)
    gen_acc_val(Opnd0)
    gen_cod(Opnd1)
    gen_acc_val(Opnd1)
    emit resta

gen_cod(exp_mult(Opnd0, Opnd1)) :
    gen_cod(Opnd0)
    gen_acc_val(Opnd0)
    gen_cod(Opnd1)
    gen_acc_val(Opnd1)
    emit mult
```

```

gen_cod(exp_div(Opnd0, Opnd1)):
    gen_cod(Opnd0)
    gen_acc_val(Opnd0)
    gen_cod(Opnd1)
    gen_acc_val(Opnd1)
    emit div

gen_cod(exp_mod(Opnd0, Opnd1)):
    gen_cod(Opnd0)
    gen_acc_val(Opnd0)
    gen_cod(Opnd1)
    gen_acc_val(Opnd1)
    emit mod

gen_cod(exp_and(Opnd0, Opnd1)):
    gen_cod(Opnd0)
    gen_acc_val(Opnd0)
    gen_cod(Opnd1)
    gen_acc_val(Opnd1)
    emit or

gen_cod(exp_or(Opnd0, Opnd1)):
    gen_cod(Opnd0)
    gen_acc_val(Opnd0)
    gen_cod(Opnd1)
    gen_acc_val(Opnd1)
    emit or

gen_cod(exp_menos(Exp)):
    gen_cod(Exp)
    gen_acc_val(Exp)
    emit menos

gen_cod(exp_not(Exp)):
    gen_cod(Exp)
    gen_acc_val(Exp)
    emit not

gen_cod(inst_index(Opnd0, Opnd1)):
    gen_cod(Opnd0)
    gen_cod(Opnd1)
    gen_acc_val(Opnd1)
    let ref!(Opnd1.tipo) = array(T,d) in
        emit apilaInt(T.tam)
    end let
    emit mult
    emit suma

gen_cod(exp_reg(Exp, id)):
    gen_cod(Exp)
    let ref!(E.tipo) = struct(Cs) in
        emit apilaInt(desplazamiento(Cs,c))
    end let
    emit suma

gen_cod(exp_ind(Exp)):
    gen_cod(Exp)
    emit apilaInd()

gen_cod(exp_true()):

```

```

        emit apilaBool("true")

gen_cod(exp_false()):
    emit apilaBool("false")

gen_cod(exp_litEnt(litEnt)):
    emit apilaInt(litEnt)

gen_cod(exp_litReal(litReal)):
    emit apilaReal(litReal)

gen_cod(exp_litCad(litCad)):
    emit apilaString(litCad)

gen_cod(exp_iden(id)):
    emit apilaInt($.viculo.dir)

gen_cod(exp_null()): noop

gen_acc_val(Exp):
    if es_designador(ref!(E)) then
        emit apilaInd()
    end if

gen_acc_id(dec_var(T, id)):
    if $.nivel = 0 then
        emit apilaInt($.vinculo.dir)
    else
        gen_acc_var($)
    end if

gen_acc_id(pval(T, id)):
    gen_acc_var($)

gen_acc_id(pref(T, id)):
    gen_acc_var($)
    emit apila_ind()

gen_acc_var(V):
    emit apilad(V.nivel)
    emit apilaInt(V.dir)
    emit suma()

gen_acc_val(E):
    if es_designador(E) then
        emit apilaInd
    end if

gen_paso_PFml(proc(id, PFml, Decs, Is), PReal):
    emit dup
    emit apilaInt(PFml.dir)
    emit suma
    gen_cod(PReal)
    if PFml = pref(T, id) v not es_designador(PReal) then
        emit desapila_ind()
    else
        let PFml = pval(T, id) in
            emit copia(T.tam)
        end let
    end if

```



```
recolecta_procs(muchas_decs(Decs, Dec)):
    recolecta_procs(muchas_decs(Decs))
    recolecta_procs(Dec)

recolecta_procs(una_dec(Dec)):
    recolecta_procs(Dec)

recolecta_procs(dec_simple(Tipo, id)): noop

recolecta_procs(dec_type(Tipo, id)): noop

recolecta_procs(dec_proc(id, PFml, Decs, Is)):
    emit_apila(sub_pendientes, $)
```

# Índice de figuras