

---

# 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*



# Índice general

<b>1. Especificación del procesamiento de vinculación</b>	<b>2</b>
<b>2. Especificación del procesamiento de comprobación de tipos</b>	<b>8</b>
<b>3. Especificación del procesamiento de asignación de espacio</b>	<b>14</b>
<b>4. Instrucciones de la máquina-p necesarias para soportar la traducción de Tiny a código-p</b>	<b>20</b>
4.0.1. Instrucciones Aritmético-Lógicas . . . . .	20
4.0.2. Instrucciones de movimiento de datos . . . . .	21
4.0.3. Instrucciones de salto . . . . .	21
4.0.4. Instrucciones de gestión de memoria dinámica . . . . .	22
4.0.5. Instrucciones de Soporte de la ejecución de procedimientos . . . . .	22
<b>5. Especificación del procesamiento de etiquetado</b>	<b>24</b>
<b>6. Especificación del procesamiento de generación de código</b>	<b>32</b>
6.1. Tipos contruidos . . . . .	32
6.2. Bloques y subprogramas . . . . .	35
<b>Índice de figuras</b>	<b>42</b>

# 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
    abreAmbito (ts)
    vincula1 (PFmls)
    vincula1 (Blo)

vincula2 (dec_proc (id ,PFmls ,Blo )):
    vincula2 (PFmls)
    vincula2 (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)):
    vincula(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)
    $.vinculo = vinculoDe (ts, Id)
    if $.vinculo == false then
        error
    end if

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 (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 = Var.tipo

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

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

tipado (tipo_array (Tipo, litEnt)) :
    tipado (Tipo)
    $.tipo = Tipo.tipo

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 = Dec_var(T,I) in
    $.tipo = T
  end let

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

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

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

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

tipado(inst_if(Exp, Blo)):
  tipado(Exp)
  tipado(Blo)
  $.tipo = ambos-ok(Exp.tipo, Blo.tipo)

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

```

```

tipado(Exp)
tipado(Blo1)
tipado(Blo2)
if Exp.tipo == ok ^ Blo1 == ok ^ Blo2 == ok then
    $.tipo = ok
else
    return error
end if

tipado(inst_while(Exp,Blo)):
    tipado(Exp)
    tipado(Blo)
    $.tipo = ambos-ok(Exp.tipo ,Blo.tipo)

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

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

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

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

tipado(inst_call(id ,PReales)):
    tipado(PReales)
    $.tipo = PReales.tipo

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 = ok
        else
            aviso-error(Opnd0.tipo ,Opnd1.tipo)
            $.tipo = error
        end if
    else
        error
        $.tipo = error
    end if

tipado(exp_menor(Opnd0,Opnd1)):
    tipado-bin(Opnd0,Opnd1)

tipado(exp_menIgual(Opnd0,Opnd1)):
    tipado-bin(Opnd0,Opnd1)

```

```
tipado(exp_mayor(Opnd0, Opnd1)):
    tipado-bin(Opnd0, Opnd1)

tipado(exp_mayIgual(Opnd0, Opnd1)):
    tipado-bin(Opnd0, Opnd1)

tipado(exp_igual(Opnd0, Opnd1)):
    tipado-bin(Opnd0, Opnd1)

tipado(exp_dist(Opnd0, Opnd1)):
    tipado-bin(Opnd0, Opnd1)

tipado(exp_sum(Opnd0, Opnd1)):
    tipado-bin(Opnd0, Opnd1)

tipado(exp_resta(Opnd0, Opnd1)):
    tipado-bin(Opnd0, Opnd1)

tipado(exp_mult(Opnd0, Opnd1)):
    tipado-bin(Opnd0, Opnd1)

tipado(exp_div(Opnd0, Opnd1)):
    tipado-bin(Opnd0, Opnd1)

tipado(exp_mod(Opnd0, Opnd1)):
    tipado-bin(Opnd0, Opnd1)

tipado(exp_and(Opnd0, Opnd1)):
    tipado-bin(Opnd0, Opnd1)

tipado(exp_or(Opnd0, Opnd1)):
    tipado-bin(Opnd0, Opnd1)

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

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

tipado(inst_index(Opnd0, Opnd1)):
    tipado-bin(Opnd0, Opnd1)

tipado(exp_reg(Exp, id)):
    tipado(Exp)
    tipado-bin(Opnd0, Opnd1)

tipado(exp_ind(Exp)):
    tipado(Exp)

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_var(T,I) in
    $.tipo = T
  end let

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

tipado(elem1(E)):
  tipado(E)
  if ref!(E.tipo) = par(T,_) then
    Acc.tipo = T
  else
    aviso-error(T)
    Acc.tipo = error
  end if

tipado(elem2(E)):
  tipado(E)
  if ref!(E.tipo) = par(_,T) then
    Acc.tipo = T
  else
    aviso-error(T)
    Acc.tipo = error
  end if

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_tipo(T',I) in
      return ref!(T')
    end let
  else
    return T
  end if

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

tipo-bin(T0,T1):
  if compatibles(T0,T1) then

```

```
        return T0
    else
        aviso-error(T0,T1)
        return error
    end if

compatibles(T1,T2):
    let T1' = ref!(T1) ^ T2' = ref!(T2) in
        if T1' == T2' then
            return true;
        elsif T1' == par(T1_a,T1_b) ^ T2' == par(T2_a,T2_b) then
            return compatibles(T1_a,T2_a) ^ compatibles(T1_b,T2_b)
        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-espacio1(si_decs(LDecs)):
    asig-espacio(Ldecs)
```

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

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

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

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

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

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

```
asig-espacio(dec_simple(Var)):
    asig-espacio(Var)
```

```
asig-espacio(dec_type(Var)):
    asig-espacio(Var)
```

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

```

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

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

asig-espacio-tipo2 (tipo_array (Tipo, litEnt)):
    si Tipo = ref(id)
        sea Tipo.vinculo = dec_type (Tipo, )
        asig-espacio-tipo (Tipo)
    $.tam = Tipo.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 = var (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 = var (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-espacio(si_pformal(LPFml)):
    asig-espacio(LPFml)

asig-espacio(no_pformal()): skip

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

asig-espacio(un_pformal(PFml))
    asig-espacio(PFml)

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

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

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)):
    asig-espacio(Exp)

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

```

```

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

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

asig-espacio ( inst__new (Exp ) ):
    asig-espacio (Exp)

asig-espacio ( inst__delete (Exp ) ):
    asig-espacio (Exp)

asig-espacio ( inst__read (Exp ) ):
    asig-espacio (Exp)

asig-espacio ( inst__write (Exp ) ):
    asig-espacio (Exp)

asig-espacio ( inst__call (id , PReales ) ):
    ant_dir = dir
    nivel = nivel + 1
    $.nivel = nivel
    dir = 0
    asigna__espacio (PReales)
    $.tam_datos = dir
    dir = ant_dir
    nivel = nivel - 1

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-tam1 (exp_reg (Exp, id )):
    ant_dir = dir
    nivel = nivel + 1
    $.nivel = nivel
    dir = 0
    asigna_espacio (Exp)
    $.tam_datos = dir
    dir = ant_dir
    nivel = nivel - 1

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-tipo1(exp_iden(id)):  
    asig-espacio-tipo1($.vinculo)  
    sea $.vinculo = dec_type(Tipo,_) en  
        $.tam = Tipo.tam  
  
asig-espacio-tipo2(exp_iden(id)): skip  
  
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 los 2 primeros argumentos 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
indireccion	Desapila el primer argumento y apila el de su indirección
indexacion	Desapila los 2 primeros argumentos y apila el valor resultante de la indexación
accesoReg	Desapila los 2 primeros argumentos y apila el resultado del acceso a registro(con el 2º argumento) sobre el primero.

#### 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
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.

#### 4.0.3. Instrucciones de salto

Cuadro 4.0.3: Instrucciones de salto

irA(d)	Salto incondicional a la dirección d.

Continúa en la siguiente página

Cuadro 4.0.3: Instrucciones de salto (Continuación)

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	

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

Cuadro 4.0.4: Instrucciones de gestión de memoria dinámica

alloc(n)	Reserva un bloque de n celdas consecutivas en el heap y apila la dirección de comienzo en la pila de evaluación.
dealloc(n)	Desapila una dirección d de la pila de evaluación y libera en el heap el bloque de n celdas consecutivas que comienza en d.

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

Cuadro 4.0.5: Instrucciones de Soporte de la ejecución de procedimientos

activa(n,t,dir)	Reserva espacio en el segmento de pila de registros de activación para ejecutar un procedimiento que tiene nivel de anidamiento <b>n</b> y tamaño de datos locales <b>t</b> . Así mismo, almacena en la zona de control de dicho registro <b>dir</b> como dirección de retorno. También almacena en dicha zona de control el valor del display de nivel <b>n</b> . Por último, apila en la pila de evaluación la dirección de comienzo de los datos en el registro creado
apilad(n)	Apila en la pila de evaluación el valor del display de nivel <b>n</b>
desapilad(n)	Desapila una dirección <b>dir</b> de la pila de evaluación en el display de nivel <b>n</b>
desactiva(n,t)	Libera el espacio ocupado por el registro de activación actual, restaurando adecuadamente el estado de la máquina. <b>n</b> indica el nivel de anidamiento del procedimiento asociado; <b>t</b> 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 <b>n</b> 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)):
  $.prim = etq
  etiquetado(Blo)
  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++
      etiquetado(Blo)
      etq+=2
      sub.sig = etq
    end let
  end while
  $.sig = etq

etiquetado(bloq(Decs, Insts)):
  $.prim = etq
  recolecta-subs(Decs)
  etiquetado(Insts)
  $.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
```

```
    etiquetado (PReales)
    $.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)
    $.sig = etq

etiquetado (exp_not (Exp)) :
    $.prim = etq
    etiquetado (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
```



# 6 | Especificación del procesamiento de generación de código

---

## 6.1. Tipos contruidos

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

gen_cod(bloq(Decs, Insts)):
    gen_cod(Insts)

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_cod(inst_if(Exp, Blo)):
    gen_cod(Exp)
    gen_cod(Blo)

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

gen_cod(inst_while(Exp, Blo)):
    gen_cod(Exp)
    gen_cod(Blo)

gen_cod(inst_new(Exp)):
    gen_cod(Exp)
```

```
gen_cod(inst_delete(Exp)):  
    gen_cod(Exp)  
  
gen_cod(inst_read(Exp)):  
    gen_cod(Exp)  
  
gen_cod(inst_write(Exp)):  
    gen_cod(Exp)  
  
gen_cod(inst_call(id, PReales)):  
    gen_cod(PReales)  
  
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)  
    emit desapilaInd($.vinculo.dir)  
  
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

```

## 6.2. Bloques y subprogramas

```
var sub_pendientes = pila_vacia()
```

```

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

gen_cod(bloq(Decs, Insts)):
    recolecta_subs(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, Insts) 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) = pointer(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) = pointer(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_cod(elem1(Exp)):
    gen_cod(Exp)

gen_cod(elem2(Exp)):
    gen_cod(Exp)
    let ref!(Exp.tipo) = Par(T_0,T_1) in
        emit apila_int(T_0.tam)
        emit suma
    end let

gen_cod(dref(Exp)):
    gen_cod(Exp)
    emit apila_ind()

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

gen_acc_id(dec_var(T,id)):
    if $.nivel = 0 then
        emit apila_int($.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 apila_int(V.dir)
    emit suma()

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

gen_paso_PFml(proc(id,PFml,Decs,Is),PReal):
    emit dup
    emit apila_int(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)):
    apila(sub_pendientes, $)
```

# Índice de figuras