

# Memoria: Procesadores de Lenguaje - Lenguaje Tiny

Fase 4: Finalización del procesador para Tiny

Grupo G03:

Burgos Sosa Rodrigo, Cassin Gina Andrea,  
Estebán Velasco Luis, Rabbia Santiago Elias

Curso 2024

## 0 Introducción

En el siguiente documento se expondrá una memoria sobre las especificaciones necesarias para finalizar la implementación del procesador de lenguaje Tiny. Se realizará:

1. Una especificación del procesamiento de vinculación.
2. Una especificación del procesamiento de comprobación de tipos.
3. Una especificación del procesamiento de asignación de espacio.
4. Una descripción del repertorio de instrucciones de la máquina-p necesario para soportar la traducción de Tiny a código-p.
5. Una especificación del procesamiento de etiquetado.
6. Una especificación del procesamiento de generación de código.

## 1 Especificación del procesamiento de vinculación

A continuación se presentará la especificación del procesamiento de vinculación. Durante este procesamiento se comprueban las reglas de ámbito del lenguaje, siendo estas dos:

- Se comprueba que, cuando se utiliza un identificador, éste haya sido previamente declarado.
- Los usos de los identificadores se vinculan con sus declaraciones.

Cada nodo de tipo `iden` tendrá un atributo `vinculo`: una referencia a nodos de tipo `Dec`.

### 1.1 Tabla de símbolos

Es un diccionario `String → Dec`, con las siguientes operaciones:

- **creaTS()**: Crea una tabla de símbolos vacía que no tiene aún ningún ámbito abierto.
- **abreAmbito(ts)**: Añade a la tabla de símbolos `ts` un nuevo ámbito, que tendrá como padre el ámbito más reciente (o  $\perp$ , si aún no se ha creado ningún ámbito).

- **contiene(ts,id)**: Comprueba si la tabla de símbolos ts contiene ya una entrada para el identificador id.
- **inserta(ts,id,dec)**: Inserta el identificador id en la tabla de símbolos ts, con la referencia al nodo dec como valor.
- **vinculoDe(ts,id)**: Recupera la referencia asociada a id en la tabla de símbolos ts. Para ello busca sucesivamente en la cadena de ámbitos, hasta que lo encuentra. Si no está, devuelve  $\perp$ .
- **cierraAmbito(ts)**: Fija en ts el ámbito actual al ámbito padre del ámbito más reciente.

## 1.2 Organización del procesamiento

- Procesar, en orden de aparición, las declaraciones en la sección de declaraciones.
- Para cada declaración:
  - Se comprueba que el identificador no esté en la tabla de símbolos (si es así, se señala error).
  - Vincular los identificadores en la expresión.
  - Asociar la declaración con el identificador en la tabla de símbolos.
- La vinculación en las expresiones compuestas se lleva a cabo vinculando en cada uno de sus operandos.
- La vinculación de las expresiones iden se lleva a cabo:
  - Comprobando que el identificador esté en la tabla de símbolos. Si no es así, se señala error.
  - Fijando el atributo vinculo al nodo asociado al identificador en la tabla de símbolos.
- Para permitir el algoritmo de compatibilidad estructural de tipos recursivos, se realizan dos pasadas en el vinculador en la sección de declaraciones.

### 1.3 Vinculador para Tiny

- `var ts // La tabla de símbolos`
- `vincula(prog(Bloq)):`  
    `ts = creaTS()`  
    `vincula(Bloq)`
- `vincula(bloq(Decs, Insts)):`  
    `abreAmbito(ts)`  
    `vincula(Decs)`  
    `vincula(Insts)`  
    `cierraAmbito(ts)`
- `vincula(si_decs(DecsAux)):`  
    `vincula1(DecsAux)`  
    `vincula2(DecsAux)`
- `vincula(no_decs()):`  
    `noop`

#### `// Primera pasada`

- `vincula1(muchas_decs(DecsAux, Dec)):`  
    `vincula1(DecsAux)`  
    `vincula1(Dec)`
- `vincula1(una_dec(Dec)):`  
    `vincula1(Dec)`
- `vincula1(dec_var(T, string)):`  
    `vincula1(T)`  
    `if contiene(ts, string) then`  
        `error`  
    `else`  
        `inserta(ts, string, $)`  
    `end if`

- `vincula1(dec_tipo(T, string)):`  
`vincula1(T)`  
`if contiene(ts, string) then`  
`error`  
`else`  
`inserta(ts, string, $)`  
`end if`
- `vincula1(dec_proc(string, ParamsF, Bloq)):`  
`if contiene(ts, string) then`  
`error`  
`else`  
`inserta(ts, string, $)`  
`end if`  
`abreAmbito(ts)`  
`vincula1(ParamsF)`  
`vincula2(ParamsF)`  
`vincula(Bloq)`  
`cierraAmbito(ts)`
- `vincula1(si_paramF(ParamsFL)):`  
`vincula1(ParamsFL)`
- `vincula1(no_paramF()):`  
`noop`
- `vincula1(muchos_paramsF(ParamsFL, Param)):`  
`vincula1(ParamsFL)`  
`vincula1(Param)`
- `vincula1(un_paramF(Param)):`  
`vincula1(Param)`

- `vincula1(param_ref(T, string))`:
 

```

vincula1(T)
if contiene(ts, string) then
  error
else
  inserta(ts, string, $)
end if

```
- `vincula1(param(T, string))`:
 

```

vincula1(T)
if contiene(ts, string) then
  error
else
  inserta(ts, string, $)
end if

```
- `vincula1(tipo_array(T, tam))`:
 

```

vincula1(T)
if tam < 0 then // Pre-tipado: El tamaño de los tipos array es
                 siempre un entero no negativo
  error
end if

```
- `vincula1(tipo_punt(T))`:
 

```

if T ≠ tipo_iden(_) then
  vincula1(T)
end if

```
- `vincula1(tipo_struct(LCampos))`:
 

```

abreAmbito(ts) // Pre-tipado: Las definiciones de tipos registro
                 no tienen campos duplicados
vincula1(LCampos)
cierraAmbito(ts)

```

- `vincula1(muchos_campos(LCampos, Campo)):`  
`vincula1(LCampos)`  
`vincula1(Campo)`
- `vincula1(un_campo(Campo)):`  
`vincula1(Campo)`
- `vincula1(campo(T, string)):`  
`vincula1(T)`  
`if contiene(ts, string) then`  
`error`  
`else`  
`inserta(ts, string, $)`  
`end if`
- `vincula1(tipo_int()):`  
`noop`
- `vincula1(tipo_real()):`  
`noop`
- `vincula1(tipo_bool()):`  
`noop`
- `vincula1(tipo_string()):`  
`noop`
- `vincula1(tipo_iden(string)):` *// Pre-tipado: Los vínculos de los nombres de tipo utilizados en las declaraciones de tipo deben ser declaraciones type*  
`$.vinculo = vinculoDe(ts,string)`  
`if $.vinculo  $\neq$  dec_tipo(.,.) then`  
`error`  
`end if`

**// Segunda pasada**

- `vincula2(muchas_decs(DecsAux, Dec)):`  
`vincula2(DecsAux)`  
`vincula2(Dec)`
- `vincula2(una_dec(Dec)):`  
`vincula2(Dec)`
- `vincula2(dec_var(T, string)):`  
`vincula2(T)`
- `vincula2(dec_tipo(T, string)):`  
`vincula2(T)`
- `vincula2(dec_proc(string, ParamsF, Bloq)):`  
`noop`
- `vincula2(si_paramF(ParamsFL)):`  
`vincula2(ParamsFL)`
- `vincula2(no_paramF()):`  
`noop`
- `vincula2(muchos_paramsF(ParamsFL, Param)):`  
`vincula2(ParamsFL)`  
`vincula2(Param)`
- `vincula2(un_paramF(Param)):`  
`vincula2(Param)`
- `vincula2(param_ref(T, string)):`  
`vincula2(T)`
- `vincula2(param(T, string)):`  
`vincula2(T)`
- `vincula2(tipo_array(T, string)):`  
`vincula2(T)`



- vincula2(tipo\_punt(T)):
  - if T = tipo\_iden(iden) then
    - T.vinculo = vinculoDe(ts,iden)
    - if T.vinculo  $\neq$  dec\_tipo(.,.) then
      - error
    - end if
  - else
    - vincula2(T)
  - end if
- vincula2(tipo\_struct(LCampos)):
  - vincula2(LCampos)
- vincula2(muchos\_campos(LCampos, Campo)):
  - vincula2(LCampos)
  - vincula2(Campo)
- vincula2(un\_campo(Campo)):
  - vincula2(Campo)
- vincula2(campo(T, string)):
  - vincula2(T)
- vincula1(tipo\_int()):
  - noop
- vincula1(tipo\_real()):
  - noop
- vincula1(tipo\_bool()):
  - noop
- vincula1(tipo\_string()):
  - noop
- vincula1(tipo\_iden(string)):
  - noop

// **Única pasada**

- `vincula(si_instr(InstAux)):`  
    `vincula(InstAux)`
- `vincula(no_instr()):`  
    `noop`
- `vincula(muchas_instr(InstsAux, Inst):`  
    `vincula(InstsAux)`  
    `vincula(Inst)`
- `vincula(una_instr(Inst)):`  
    `vincula(Inst)`
- `vincula(instr_expr(Exp)):`  
    `vincula(Exp)`
- `vincula(instr_if(Exp, Bloq)):`  
    `vincula(Exp)`  
    `vincula(Bloq)`
- `vincula(instr_if_else(Exp, Bloq1, Bloq2)):`  
    `vincula(Exp)`  
    `vincula(Bloq1)`  
    `vincula(Bloq2)`
- `vincula(instr_while(Exp, Bloq)):`  
    `vincula(Exp)`  
    `vincula(Bloq)`
- `vincula(instr_read(Exp)):`  
    `vincula(Exp)`
- `vincula(instr_write(Exp)):`  
    `vincula(Exp)`
- `vincula(instr_nl()):`  
    `noop`

- `vincula(instr_new(Exp)):`  
`vincula(Exp)`
- `vincula(instr_del(Exp)):`  
`vincula(Exp)`
- `vincula(instr_call(string, ParamsR)):`  
`vincula(ParamsR)`  
`$.vinculo = vinculoDe(ts,string)`  
`if $.vinculo ==  $\perp$  then`  
`error`  
`end if`
- `vincula(instr_bloque(Bloq)):`  
`vincula(Bloq)`
- `vincula(si_paramsR(ParamsRL)):`  
`vincula(ParamsRL)`
- `vincula(no_paramsR()):`  
`noop`
- `vincula(muchos_paramsR(ParamsRL, Exp)):`  
`vincula(ParamsRL)`  
`vincula(Exp)`
- `vincula(un_paramsR(Exp)):`  
`vincula(Exp)`
- `vincula(asig(Exp1, Exp2)):`  
`vincula(Exp1)`  
`vincula(Exp2)`
- `vincula(menor(Exp1, Exp2)):`  
`vincula(Exp1)`  
`vincula(Exp2)`

- $\text{vincula}(\text{mayor}(\text{Exp1}, \text{Exp2}))$ :  
 $\text{vincula}(\text{Exp1})$   
 $\text{vincula}(\text{Exp2})$
- $\text{vincula}(\text{menor\_igual}(\text{Exp1}, \text{Exp2}))$ :  
 $\text{vincula}(\text{Exp1})$   
 $\text{vincula}(\text{Exp2})$
- $\text{vincula}(\text{mayor\_igual}(\text{Exp1}, \text{Exp2}))$ :  
 $\text{vincula}(\text{Exp1})$   
 $\text{vincula}(\text{Exp2})$
- $\text{vincula}(\text{igual}(\text{Exp1}, \text{Exp2}))$ :  
 $\text{vincula}(\text{Exp1})$   
 $\text{vincula}(\text{Exp2})$
- $\text{vincula}(\text{no\_igual}(\text{Exp1}, \text{Exp2}))$ :  
 $\text{vincula}(\text{Exp1})$   
 $\text{vincula}(\text{Exp2})$
- $\text{vincula}(\text{suma}(\text{Exp1}, \text{Exp2}))$ :  
 $\text{vincula}(\text{Exp1})$   
 $\text{vincula}(\text{Exp2})$
- $\text{vincula}(\text{resta}(\text{Exp1}, \text{Exp2}))$ :  
 $\text{vincula}(\text{Exp1})$   
 $\text{vincula}(\text{Exp2})$
- $\text{vincula}(\text{and}(\text{Exp1}, \text{Exp2}))$ :  
 $\text{vincula}(\text{Exp1})$   
 $\text{vincula}(\text{Exp2})$
- $\text{vincula}(\text{or}(\text{Exp1}, \text{Exp2}))$ :  
 $\text{vincula}(\text{Exp1})$   
 $\text{vincula}(\text{Exp2})$

- `vincula(mult(Exp1, Exp2)):`  
`vincula(Exp1)`  
`vincula(Exp2)`
- `vincula(div(Exp1, Exp2)):`  
`vincula(Exp1)`  
`vincula(Exp2)`
- `vincula(mod(Exp1, Exp2)):`  
`vincula(Exp1)`  
`vincula(Exp2)`
- `vincula(negativo(Exp)):`  
`vincula(Exp)`
- `vincula(not(Exp)):`  
`vincula(Exp)`
- `vincula(index(Exp1, Exp2)):`  
`vincula(Exp1)`  
`vincula(Exp2)`
- `vincula(acceso(Exp, string)):`  
`vincula(Exp)`
- `vincula(indireccion(Exp)):`  
`vincula(Exp)`
- `vincula(lit_ent(string)):`  
`noop`
- `vincula(lit_real(string)):`  
`noop`
- `vincula(true()):`  
`noop`
- `vincula(false()):`  
`noop`

- `vincula(lit_cadena(string)):`  
`noop`
- `vincula(iden(string)):`  
`$.vinculo = vinculoDe(ts,string)`  
`if $.vinculo ==  $\perp$  then`  
`error`  
`end if`
- `vincula(null()):`  
`noop`

## 2 Especificación del procesamiento de comprobación de tipos

A continuación se presentará la especificación del procesamiento de comprobación de tipos. Durante este procesamiento se comprueban las reglas de tipado del lenguaje:

- Se comprueba que las distintas construcciones del lenguaje estén correctamente tipadas.
- Se asocian tipos con estas construcciones.

Tiny es un lenguaje fuertemente tipado. Se crean dos nuevos tipos: `ok` y `error` para la gestión de errores.

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

```

    end if
    if T1 == error then
        error
    end if

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

• ref!(T): // sigue la cadena de vínculos entre nombres de tipos sinónimos
    if T = tipo_iden(iden) then
        let T.vinculo = dec_tipo(T',iden) in
            return ref!(T')
        end let
    else
        return T
    end if

• es-designador(E) // determina cuándo una expresión es o no un designador
    return E == iden(_) || E == acceso(,-) || E == index(,-) || E
    == indireccion(_)

• son_unificables(muchos_campos(LCampos1, Campo(T1, Iden1)), muchos_campos(LCampos2, Campo(T2, Iden2)), esParamRef):
    if son_unificables(T1, T2, esParamRef) then
        return son_unificables(LCampos1, LCampos2, esParamRef)
    else
        return false
    end if

• son_unificables(muchos_campos(LCampos1, Campo1), un_campo(Campo2), esParamRef):
    return false

```

- `son_unificables(un_campo(Campo1), muchos_campos(LCampos2, Campo2), esParamRef):`  
`return false`
- `son_unificables(un_campo(Campo(T1, Iden1)), un_campo(Campo(T2, Iden2)), esParamRef):`  
`return son_unificables(T1, T2)`
- `son_unificables(T1, T2, esParamRef):`  
`if (T1 = T2)  $\notin$   $\Theta$  then`  
 `$\Theta = \Theta \cup \{T1 = T2\}$`   
`return unificables(T1, T2, esParamRef)`  
`else`  
`return true`  
`end if`
- `unificables(T1, T2, esParamRef):`  
`let T1' = ref!(T1), T2' = ref!(T2) in`  
`if T1' == T2' then`  
`return true`  
`else if T1' == tipo_real(n1) && T2' == tipo_int(n2) then // se`  
`asigna un int a un real`  
`return !esParamRef`  
`else if T1' == tipo_punt(T1a) && T2' == null(n2) then // se`  
`asigna null a un puntero`  
`return true`  
`else if T1' == tipo_array(T1a, n1) && T2' == tipo_array(T2a,`  
`n1) then`  
`return n1 == n2 && son_unificables(T1a, T2a, esParam-`  
`Ref)`  
`else if T1' == tipo_struct(LCampos1) && T2' == tipo_struct(LCampos2)`  
`then`  
`return son_unificables(LCampos1, LCampos2, esParam-`  
`Ref)`



```

    else if T1' == tipo_punt(T1a) && T2' == tipo_punt(T2a) then
        return son_unificables(T1a,T2a, esParamRef)
    else
        return false
    end if
end let

• compatibles(T1,T2):
     $\Theta = \{T1 = T2\}$ 
    return unificables(T1,T2, false)

• compatibles(T1, T2, esParamRef):
     $\Theta = \{T1 = T2\}$ 
    return unificables(T1,T2, true)

• tipado(prog(Bloq)):
    tipado(Bloq)
    $.tipo = Bloq.tipo

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

• tipado(si_decs(DecsAux)):
    tipado(DecsAux)
    $.tipo = DecsAux.tipo

• tipado(no_decs()):
    $.tipo = ok

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

```

- `tipado(una_dec(Dec)):`  
`tipado(Dec)`  
`$.tipo = Dec.tipo`
- `tipado(dec_var(T, Iden)):`  
`$.tipo = ok`
- `tipado(dec_tipo(T, Iden)):`  
`$.tipo = ok`
- `tipado(dec_proc(Iden, ParamsF, Bloq)):`  
`tipado(Bloq)`  
`$.tipo = Bloq.tipo`
- `tipado(si_instr(InstsAux)):`  
`tipado(InstsAux)`  
`$.tipo = InstsAux.tipo`
- `tipado(no_instr()):`  
`$.tipo = ok`
- `tipado(muchas_instr(InstsAux, Inst)):`  
`tipado(InstsAux)`  
`tipado(Inst)`  
`$.tipo = ambos-ok(InstsAux.tipo, Inst.tipo)`
- `tipado(una_instr(Inst)):`  
`tipado(Inst)`  
`$.tipo = Inst.tipo`
- `tipado(instr_expr(Exp)):`  
`tipado(Exp)`  
`if Exp.tipo == error then`  
`$.tipo = error`  
`else`  
`aviso-error(Exp.tipo)`

```

        $.tipo = ok
    end if

    • tipado(instr_if(Exp, Bloq)):
        tipado(Exp)
        tipado(Bloq)
        if ref!(Exp.tipo) == tipo_bool && Bloq.tipo == ok then
            $.tipo = ok
        else
            $.tipo = error
        end if

    • tipado(instr_if_else(Exp, Bloq1, Bloq2)):
        tipado(Exp)
        tipado(Bloq1)
        tipado(Bloq2)
        if ref!(Exp.tipo) == tipo_bool && Bloq1.tipo == ok && Bloq2.tipo
        == ok then
            $.tipo = ok
        else
            aviso-error(Exp.tipo)
            $.tipo = error
        end if

    • tipado(instr_while(Exp, Bloq)):
        tipado(Exp)
        tipado(Bloq)
        if ref!(Exp.tipo) == tipo_bool && Bloq.tipo == ok then
            $.tipo = ok
        else
            aviso-error(Exp.tipo)
            $.tipo = error
        end if

```

- `tipado(instr_read(Exp)):`  
`tipado(Exp)`  
`t = ref!(Exp.tipo)`  
`if (t == tipo_int || t == tipo_real || t == tipo_string) && es-`  
`designador(Exp) then`  
`$.tipo = ok`  
`else`  
`aviso-error(Exp.tipo)`  
`$.tipo = error`  
`end if`
- `tipado(instr_write(Exp)):`  
`tipado(Exp)`  
`t = ref!(Exp.tipo)`  
`if t == tipo_int || t == tipo_real || t == tipo_string || t ==`  
`tipo_bool then`  
`$.tipo = ok`  
`else`  
`$.tipo = error`  
`end if`
- `tipado(instr_nl()):`  
`$.tipo = ok`
- `tipado(instr_new(Exp)):`  
`tipado(Exp)`  
`if ref!(Exp.tipo) == tipo_punt then`  
`$.tipo = ok`  
`else`  
`aviso-error(Exp.tipo)`  
`$.tipo = error`  
`end if`

- `tipado(instr_del(Exp)):`  
`tipado(Exp)`  
`if ref!(Exp.tipo) == tipo_punt then`  
`$.tipo = ok`  
`else`  
`aviso-error(Exp.tipo)`  
`$.tipo = error`  
`end if`
- `llamadas_compatibles(si_paramF(ParamsFL), si_paramsR(ParamsRL)):`  
`return llamadas_compatibles(ParamsFL, ParamsRL)`
- `llamadas_compatibles(no_paramF(), no_paramsR()):`  
`return ok`
- `llamadas_compatibles(muchos_paramsF(ParamsFL, param(T, Iden)),  
muchos_paramsR(ParamsRL, Exp)):`  
`if es-designador(Exp) && compatibles(Exp.tipo, T) then`  
`return llamadas_compatibles(ParamsFL, ParamsRL)`  
`else`  
`aviso-error(Exp)`  
`return error`  
`end if`
- `llamadas_compatibles(muchos_paramsF(ParamsFL, param_ref(T, Iden)),  
muchos_paramsR(ParamsRL, Exp)):`  
`if compatibles(Exp.tipo, T) then`  
`return llamadas_compatibles(ParamsFL, ParamsRL)`  
`else`  
`aviso-error(Exp)`  
`return error`  
`end if`

- `llamadas_compatibles(un_paramF(param(T, Iden)), un_paramsR(Exp)):`  
     if `compatibles(T, Exp.tipo)` then  
         return ok  
     else  
         aviso-error(Exp)  
         return error  
     end if
- `llamadas_compatibles(un_paramF(param_ref(T, Iden)), un_paramsR(Exp)):`  
     if `!es-designador(Exp)` then  
         error  
     if `compatibles(T, Exp.tipo, true)` then  
         return ok  
     else  
         aviso-error(Exp)  
         return error  
     end if
- `tipado(instr_call(Iden, ParamsR)):`  
     `tipado(paramsR)` // *primero tipamos los parámetros R*  
     if `Iden.vinculo ≠ dec_proc(.,.)` then  
         aviso-error(Iden)  
         \$.tipo = error  
         return  
     let `Iden.vinculo = dec_proc(Iden, ParamsF, Bloq)` in  
         if `num_elems(paramsF) == num_elems(ParamsR)` then // *luego verificamos cantidad*  
             \$.tipo = `llamadas_compatibles(ParamsF, ParamsR)` // *por último, compatibilidad*  
         else  
             \$.tipo = error  
         end if  
     end let

- `tipado(no_paramsR()):`  
`$.tipo = ok`
- `tipado(si_paramsR(ParamsRL)):`  
`tipado(ParamsRL)`  
`$.tipo = ParamsRL.tipo`
- `tipado(muchos_paramsR(ParamsRL, Exp)):`  
`tipado(ParamsRL)`  
`tipado(Exp)`  
`if Exp.tipo == error then`  
`$.tipo = error`  
`else`  
`$.tipo = ParamsRL.tipo`  
`end if`
- `tipado(un_paramsR(Exp)):`  
`tipado(Exp)`  
`if Exp.tipo == error then`  
`$.tipo = error`  
`else`  
`$.tipo = ok`  
`end if`
- `num_elems(si_paramF(ParamsFL)):`  
`num_elems(ParamsFL)`
- `num_elems(no_paramF()):`  
`return 0`
- `num_elems(muchos_paramsF(ParamsFL, _)):`  
`return 1 + num_elems(ParamsFL)`
- `num_elems(un_paramF(_)):`  
`return 1`

- num\_elems(si\_paramsR(ParamsRL)):
  - num\_elems(ParamsRL)
- num\_elems(no\_paramsR()):
  - return 0
- num\_elems(muchos\_paramsR(ParamsRL,-)):
  - return 1 + num\_elems(ParamsRL)
- num\_elems(un\_paramsR(-)):
  - return 1
- tipado(instr\_bloque(Bloq)):
  - tipado(Bloq)
  - \$.tipo = Bloq.tipo
- tipado(asig(ExpI,ExpD)):
  - tipado(ExpI)
  - tipado(ExpD)
  - if es-designador(ExpI) then
    - if compatibles(ExpI.tipo,ExpD.tipo) then
      - \$.tipo = ExpI.tipo
    - else
      - aviso-error(\$)
      - \$.tipo = error
  - end if
  - else
    - aviso-error(\$) // la parte izq. debe ser un designador
    - \$.tipo = error
  - end if
- tipado-bin-comp(E1, E2, E):
  - tipado(E1)
  - tipado(E2)
  - t1 = ref!(E1.tipo)



```

    t2 = ref!(E2.tipo)
    if ((t1 == tipo_int || t1 == tipo_real) && (t2 == tipo_int || t2
== tipo_real)) || (t1 == tipo_bool && t2 == tipo_bool) || (t1 ==
tipo_string && t2 == tipo_string) then
        E.tipo = tipo_bool
    else
        aviso-error(E)
        E.tipo = error
    end if

```

- tipado(menor(Exp1, Exp2)):

```

        tipado-bin-comp(Exp1, Exp2, $)

```
- tipado(mayor(Exp1, Exp2)):

```

        tipado-bin-comp(Exp1, Exp2, $)

```
- tipado(menor\_igual(Exp1, Exp2)):

```

        tipado-bin-comp(Exp1, Exp2, $)

```
- tipado(mayor\_igual(Exp1, Exp2)):

```

        tipado-bin-comp(Exp1, Exp2, $)

```
- tipado-bin-igualdad(E1, E2, E):

```

        tipado(E1)
        tipado(E2)
        t1 = ref!(E1.tipo)
        t2 = ref!(E2.tipo)
        if ((t1 == tipo_int || t1 == tipo_real) && (t2 == tipo_int || t2
== tipo_real)) || (t1 == tipo_bool && t2 == tipo_bool) || (t1 ==
tipo_string && t2 == tipo_string) || ((t1 == tipo_punt || t1 == null)
&& (t2 == tipo_punt || t2 == null)) then
            E.tipo = tipo_bool
        else
            aviso-error(E)
            E.tipo = error
        end if

```

- `tipado(igual(Exp1, Exp2)):`  
`tipado-bin-igualdad(Exp1, Exp2, $)`
- `tipado(no_igual(Exp1, Exp2)):`  
`tipado-bin-igualdad(Exp1, Exp2, $)`
- `tipado-bin-arit(E1, E2, E):`  
`tipado(E1)`  
`tipado(E2)`  
`t1 = ref!(E1.tipo)`  
`t2 = ref!(E2.tipo)`  
`if t1 == tipo_int && t2 == tipo_int then`  
`E.tipo = tipo_int`  
`else if (t1 == tipo_int || t1 == tipo_real) && (t2 == tipo_int ||`  
`t2 == tipo_real) then`  
`E.tipo = tipo_real`  
`else`  
`aviso-error(E)`  
`E.tipo = error`  
`end if`
- `tipado(suma(Exp1, Exp2)):`  
`tipado-bin-arit(Exp1, Exp2, $)`
- `tipado(resta(Exp1, Exp2)):`  
`tipado-bin-arit(Exp1, Exp2, $)`
- `tipado(mul(Exp1, Exp2)):`  
`tipado-bin-arit(Exp1, Exp2, $)`
- `tipado(div(Exp1, Exp2)):`  
`tipado-bin-arit(Exp1, Exp2, $)`
- `tipado-bin-logi(E1, E2, E):`  
`tipado(E1)`  
`tipado(E2)`

```

t1 = ref!(E1.tipo)
t2 = ref!(E2.tipo)
if t1 == tipo_bool && t2 == tipo_bool then
    E.tipo = tipo_bool
else
    aviso-error(E)
    E.tipo = error
end if

```

- tipado(and(Exp1, Exp2)):

```

tipado-bin-logi(Exp1, Exp2, $)

```
- tipado(or(Exp1, Exp2)):

```

tipado-bin-logi(Exp1, Exp2, $)

```
- tipado(mod(Exp1, Exp2)):

```

tipado(Exp1)
tipado(Exp2)
t1 = ref!(Exp1.tipo)
t2 = ref!(Exp2.tipo)
if t1 == tipo_int && t2 == tipo_int then
    $.tipo = tipo_int
else
    aviso-error($)
    $.tipo = error
end if

```
- tipado(negativo(Exp)):

```

tipado(Exp)
t = ref!(Exp.tipo)
if t == tipo_int || t == tipo_real then
    $.tipo = t
else

```

```

        aviso-error(t)
        $.tipo = error
    end if

    • tipado(not(Exp)):
        tipado(Exp)
        if ref!(Exp.tipo) == tipo_bool then
            $.tipo = tipo_bool
        else
            aviso-error(t)
            $.tipo = error
        end if

    • tipado(index(Exp, LitEnt)):
        tipado(Exp)
        tipado(LitEnt)
        if ref!(Exp.tipo) == tipo_array(T, LitEnt) && ref!(LitEnt.tipo)
        == tipo_int then
            $.tipo = T
        else
            aviso-error(Exp)
            $.tipo = error
        end if

    • tipado(acceso(Exp, Iden)):
        tipado(Exp)
        if ref!(Exp.tipo) == tipo_struct(LCampos) then
            $.tipo = esCampoDe(Iden, LCampos)
        else
            aviso-error(Exp.tipo)
            $.tipo = error
        end if

```

- `esCampoDe(Iden, muchos_campos(LCampos, Campo))`:
 

```

        t = esCampoDe(Iden, Campo)
        if t == error then
          return esCampoDe(Iden, LCampos)
        else
          return t
        end if
      
```
- `esCampoDe(Iden, un_campo(LCampos, Campo))`:
 

```

        return esCampoDe(Iden, Campo)
      
```
- `esCampoDe(Iden1(N1), campo(T, Iden2(N2)))`:
 

```

        if N1 == N2 then
          return T
        else
          return error
        end if
      
```
- `tipado(indireccion(Exp))`:
 

```

        tipado(Exp)
        if Exp.tipo == tipo_punt(T) then
          $.tipo = T
        else
          aviso-error(Exp.tipo)
          $.tipo = error
        end if
      
```
- `tipado(lit_ent(N))`:
 

```

        $.tipo = tipo_int
      
```
- `tipado(lit_real(N))`:
 

```

        $.tipo = tipo_real
      
```
- `tipado(true(N))`:
 

```

        $.tipo = tipo_bool
      
```

- `tipado(false(N))`:  
`$.tipo = tipo_bool`
- `tipado(lit_cadena(N))`:  
`$.tipo = tipo_string`
- `tipado(iden(N))`:  
`if $.vinculo == Dec_var(T, I) then`  
`let $.vinculo = Dec_var(T, I) in`  
`$.tipo = T`  
`end let`  
`else if $.vinculo == param_ref(T, I) then`  
`let $.vinculo = param_ref(T, I) in`  
`$.tipo = T`  
`end let`  
`else if $.vinculo == param(T, I) then`  
`let $.vinculo = param(T, I) in`  
`$.tipo = T`  
`end let`  
`else`  
`error`  
`end if`
- `tipado(null())`:  
`$.tipo = null`

### 3 Especificación del proceso de asignación de espacio

Dado que, para acceder a los objetos designados, es necesario computar direcciones, será necesario también equipar la máquina con instrucciones que funcionen con direccionamiento indirecto. A continuación se definirá la asignación de espacio. Esta se hará en dos pasadas, como el proceso de vinculación.

- `var dir = 0 // contador de direcciones`
- `var max_dir = 0 // mantiene la máxima dirección asignada`
- `var nivel = 0`
- `asig-espacio(prog(Bloq)):`  
`asig-espacio(Bloq)`
- `asig-espacio(bloq(Decs, Insts)):`  
`dir_ant = dir`  
`asig-espacio1(Decs)`  
`asig-espacio2(Decs)`  
`asig-espacio(Insts)`  
`dir = dir_ant`
- `inc_dir(inc): // se incrementa dir y se mantiene max_dir`  
`dir += inc`  
`if dir > max_dir then`  
`max_dir = dir`

#### **// Primera pasada**

- `asig-espacio1(si_decs(DecsAux)):`  
`asig-espacio1(DecsAux)`
- `asig-espacio1(no_decs()):`  
`noop`
- `asig-espacio1(muchas_decs(DecsAux, Dec)):`  
`asig-espacio1(DecsAux)`  
`asig-espacio1(Dec)`
- `asig-espacio1(una_dec(Dec)):`  
`asig-espacio1(Dec)`

- `asig-espacio1(dec_var(T, string)):`  
`asig-tam1(T)`  
`$.dir = dir`  
`$.nivel = nivel`  
`inc_dir(T.tam)`
- `asig-espacio1(dec_tipo(T, string)):`  
`asig-tam1(T)`
- `asig-espacio1(dec_proc(string, ParamsF, Bloq)):`  
`dir_ant = dir`  
`max_dir_ant = max_dir`  
`nivel++`  
`$.nivel = nivel`  
`dir = 0`  
`max_dir = 0`  
`asig-espacio1(ParamsF)`  
`asig-espacio2(ParamsF)`  
`asig-espacio(Bloq)`  
`$.tam = max_dir`  
`dir = dir_ant`  
`max_dir = max_dir_ant`  
`nivel--`
- `asig-espacio1(si_paramF(ParamsFL)):`  
`asig-espacio1(ParamsFL)`
- `asig-espacio1(no_paramF()):`  
`noop`
- `asig-espacio1(muchos_paramsF(ParamsFL, Param)):`  
`asig-espacio1(ParamsFL)`  
`asig-espacio1(Param)`



- `asig-espacio1(un_paramF(Param)):`  
`asig-espacio1(Param)`
- `asig-espacio1(param_ref(T, string)):`  
`asig-tam1(T)`  
`$.nivel = nivel`  
`$.dir = dir`  
`inc_dir(T.tam)`
- `asig-espacio1(param_no_ref(T, string)):`  
`asig-tam1(T)`  
`$.nivel = nivel`  
`$.dir = dir`  
`inc_dir(T.tam)`
- `asig-tam1(tipo_array(T, tam)):`  
`asig-tam1(T)`  
`$.tam = T.tam * tam`
- `asig-tam1(tipo_punt(T)):`  
`if T  $\neq$  tipo_iden(iden) then`  
`asig-tam1(T)`  
`$.tam = 1`
- `asig-tam1(tipo_struct(LCampos)):`  
`dir_ant = dir`  
`dir = 0`  
`asig-espacio1(LCampos)`  
`$.tam = dir`  
`dir = dir_ant`
- `asig-espacio1(muchos_campos(LCampos, Campo)):`  
`asig-espacio1(LCampos)`  
`asig-espacio1(Campo)`

- `asig-espacio1(un_campo(Campo)):`  
`asig-espacio1(Campo)`
- `asig-espacio1(campo(T, string)):`  
`$.desp = dir`  
`asig.tam1(T)`  
`dir += T.tam`
- `asig-tam1(tipo_int()):`  
`$.tam = 1`
- `asig-tam1(tipo_real()):`  
`$.tam = 1`
- `asig-tam1(tipo_bool()):`  
`$.tam = 1`
- `asig-tam1(tipo_string()):`  
`$.tam = 1`
- `asig-tam1(tipo_iden(string)):`  
`$.tam = 1`

## // Segunda pasada

- `asig-espacio2(si_decs(DecsAux)):`  
`asig-espacio2(DecsAux)`
- `asig-espacio2(no_decs()):`  
`noop`
- `asig-espacio2(muchas_decs(DecsAux, Dec)):`  
`asig-espacio2(DecsAux)`  
`asig-espacio2(Dec)`
- `asig-espacio2(una_dec(Dec)):`  
`asig-espacio2(Dec)`
- `asig-espacio2(dec_var(T, string)):`  
`asig-tam2(T)`

- `asig-espacio2(dec_tipo(T, string)):`  
`asig-tam2(T)`
- `asig-espacio2(dec_proc(string, ParamsF, Bloq)):`  
`noop`
- `asig-espacio2(si_paramF(ParamsFL)):`  
`asig-espacio2(ParamsFL)`
- `asig-espacio2(no_paramF()):`  
`noop`
- `asig-espacio2(muchos_paramsF(ParamsFL, Param)):`  
`asig-espacio2(ParamsFL)`  
`asig-espacio2(Param)`
- `asig-espacio2(un_paramF(Param)):`  
`asig-espacio2(Param)`
- `asig-espacio2(param_ref(T, string)):`  
`asig-tam2(T)`
- `asig-espacio2(param(T, string)):`  
`asig-tam2(T)`
- `asig-tam2(tipo_array(T, string)):`  
`asig-tam2(T)`
- `asig-tam2(tipo_punt(T)):`  
`if T = tipo_iden(iden) then`  
`let T.vinculo = dec_tipo(T',id) in`  
`T.tam = T'.tam`  
`else`  
`asig-tam2(T)`  
`end if`
- `asig-tam2(tipo_struct(LCampos)):`  
`asig-espacio2(LCampos)`

- `asig-espacio2(muchos_campos(LCampos, Campo)):`  
`asig-espacio2(LCampos)`  
`asig-espacio2(Campo)`
- `asig-espacio2(un_campo(Campo)):`  
`asig-espacio2(Campo)`
- `asig-espacio2(campo(T, string)):`  
`asig.tam2(T)`
- `asig-tam2(tipo_int()):`  
`noop`
- `asig-tam2(tipo_real()):`  
`noop`
- `asig-tam2(tipo_bool()):`  
`noop`
- `asig-tam2(tipo_string()):`  
`noop`
- `asig-tam2(tipo_iden(string)):`  
`noop`

## // **Única pasada**

- `asig-espacio(si_instr(InstsAux)):`  
`asig-espacio(InstsAux)`
- `asig-espacio(no_instr()):`  
`noop`
- `asig-espacio(muchas_instr(InstsAux, Inst)):`  
`asig-espacio(InstsAux)`  
`asig-espacio(Inst)`
- `asig-espacio(una_instr(Inst)):`  
`asig-espacio(Inst)`

- `asig-espacio(instr_expr(Exp)):`  
`noop`
- `asig-espacio(instr_if(Exp, Bloq)):`  
`asig-espacio(Bloq)`
- `asig-espacio(instr_if_else(Exp, Bloq1, Bloq2)):`  
`asig-espacio(Bloq1)`  
`asig-espacio(Bloq2)`
- `asig-espacio(instr_while(Exp, Bloq)):`  
`asig-espacio(Bloq)`
- `asig-espacio(instr_read(Exp)):`  
`noop`
- `asig-espacio(instr_write(Exp)):`  
`noop`
- `asig-espacio(instr_nl()):`  
`noop`
- `asig-espacio(instr_new(Exp)):`  
`noop`
- `asig-espacio(instr_del(Exp)):`  
`noop`
- `asig-espacio(instr_call(string, ParamsR)):`  
`asig-espacio(ParamsR)`
- `asig-espacio(instr_bloque(Bloq)):`  
`asig-espacio(Bloq)`
- `asig-espacio(si_paramsR(ParamsRL)):`  
`asig-espacio(ParamsRL)`
- `asig-espacio(no_paramsR()):`  
`noop`

- `asig-espacio(muchos_paramsR(ParamsRL, Exp)):`  
`asig-espacio(ParamsRL)`
- `asig-espacio(un_paramsR(Exp)):`  
`noop`
- `asig-espacio(asig(Exp1, Exp2)):`  
`noop`
- `asig-espacio(menor(Exp1, Exp2)):`  
`noop`
- `asig-espacio(mayor(Exp1, Exp2)):`  
`noop`
- `asig-espacio(menor_igual(Exp1, Exp2)):`  
`noop`
- `asig-espacio(mayor_igual(Exp1, Exp2)):`  
`noop`
- `asig-espacio(igual(Exp1, Exp2)):`  
`noop`
- `asig-espacio(no_igual(Exp1, Exp2)):`  
`noop`
- `asig-espacio(suma(Exp1, Exp2)):`  
`noop`
- `asig-espacio(resta(Exp1, Exp2)):`  
`noop`
- `asig-espacio(and(Exp1, Exp2)):`  
`noop`
- `asig-espacio(or(Exp1, Exp2)):`  
`noop`

- `asig-espacio(mult(Exp1, Exp2)):`  
    `noop`
- `asig-espacio(div(Exp1, Exp2)):`  
    `noop`
- `asig-espacio(mod(Exp1, Exp2)):`  
    `noop`
- `asig-espacio(negativo(Exp)):`  
    `noop`
- `asig-espacio(not(Exp)):`  
    `noop`
- `asig-espacio(index(Exp1, Exp2)):`  
    `noop`
- `asig-espacio(acceso(Exp, string)):`  
    `noop`
- `asig-espacio(indireccion(Exp)):`  
    `noop`
- `asig-espacio(lit_ent(string)):`  
    `noop`
- `asig-espacio(lit_real(string)):`  
    `noop`
- `asig-espacio(true()):`  
    `noop`
- `asig-espacio(false()):`  
    `noop`
- `asig-espacio(lit_cadena(string)):`  
    `noop`

- `asig-espacio(iden(string)):`  
noop
- `asig-espacio(null()):`  
noop

## 4 Descripción del repertorio de instrucciones de la máquina-p necesarias para soportar la traducción de Tiny a código-p

### Instrucciones de movimientos de datos

- **`apila_int(int val):`** usado para apilar en la pila de evaluación literales para ser usados en las expresiones y los valores necesarios para calcular el tamaño en memoria de algunos tipos.
- **`apila_real(int val):`** usado para apilar en la pila de evaluación valores reales.
- **`apila_bool(boolean val):`** usado para apilar en la pila de evaluación valores booleanos verdaderos y falsos.
- **`apila_string(string val):`** usado para aplicar en la pila de evaluación el valor *val* de tipo string.
- **`apilad(int nivel):`** usado para apilar en la pila de evaluación el valor del display de nivel *nivel*.
- **`apila_ind():`** usado para llevar el valor de un designador a la cima de la pila.
- **`desapila_ind():`** desapila un valor *v* y una dirección *d* de la pila de evaluación (primero *v*, después *d*), y actualiza el contenido de la celda *d* en la memoria de datos a *v*.
- **`copia(int tam):`** al terminar las instrucciones de paso de parámetro, puede darse el caso donde en la cima de la pila de evaluación se encuentre la dirección del objeto, entonces solo habrá que copiar dicho valor a la dirección de comienzo del parámetro formal, que es la siguiente de la pila. *tam* es el tamaño del tipo que tiene que copiarse.
- **`desapilad(int nivel):`** desapila una dirección *d* de la pila de evaluación en el display de nivel *nivel*.



### Instrucciones de salto

- **ir\_a(int dir):** usado en la instrucción de bucle while para volver al comienzo del bucle y en la instrucción de llamada a una función (call) para ir al comienzo de sus instrucciones.
- **ir\_f(int dir):** usado en las instrucciones con expresiones condicionales para saltar a la instrucción *dir* si el valor en la cima de la pila es falso.
- **ir\_ind():** usado para desapilar una dirección de la pila de evaluación y realizar un salto a la misma. Vuelve a la instrucción siguiente de la llamada realizada con anterioridad a un subprograma.

### Gestión de memoria dinámica

- **alloc(int tam):** reservar el número especificado por *tam* de celdas en memoria dinámica (instrucción new).
- **dealloc(int tam):** considera libres *tam* celdas de memoria dinámica, especificadas por la dirección en la cima de la pila de evaluación (instrucción delete).

### Soporte a la ejecución de procedimientos

- **activa(int nivel, int tam, int dirretorno):** reserva espacio en el segmento de pila de registros de activación para ejecutar un procedimiento que tiene nivel de anidamiento *nivel* y tamaño de datos locales *tam*.
- **desactiva(int nivel, int tam):** libera el espacio ocupado por el registro de activación actual, restaurando adecuadamente el estado de la máquina.
- **dup():** consulta el valor *v* de la cima de la pila de evaluación, y apila de nuevo dicho valor.
- **stop():** detiene la máquina. Para tras traducir el código del programa principal y da paso a la traducción de los subprogramas apilados.
- **emit(Instruccion i):** emite una instrucción de la máquina virtual.
- **desplazamiento(LCampos Cs, string tam):** recupera el desplazamiento de un campo de una lista de campos.

## 5 Especificación del proceso de etiquetado

Cada vez que se genera una instrucción de salto, es necesario conocer a qué dirección saltar. Se realiza entonces un procesamiento previo que, para cada nodo del AST que dé lugar a código, fije la dirección de la primera instrucción que conforma dicho código, así como la dirección de la instrucción que sigue a la última instrucción (instrucción siguiente). Durante el etiquetado se cuentan las instrucciones que se van a emitir posteriormente, durante la generación.

- `var sub_pendientes = pila-vacia()`
- `var etq = 0 // cuenta la dirección de la instrucción que habrá de emitirse actualmente`
- `etiquetado(prog(Bloq)):`

```
$.prim = etq
etiquetado(Bloq)
etq++
while !es-vacia(sub_pendientes)
  sub = cima(sub_pendientes)
  desapila_ind(sub_pendientes)
  let sub = dec_proc(string,ParamsFL,Bloq) in
    sub.prim = etq
    etq++
    etiquetado(Bloq)
    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\_instr(InstAux)):

```

    etiquetado(InstAux)

```
- etiquetado(no\_instr()):

```

    noop

```
- etiquetado(muchas\_instr(InstsAux, Inst)):

```

    $.prim = etq
    etiquetado(InstsAux)
    etiquetado(Inst)
    $.sig = etq

```
- etiquetado(una\_instr(Inst)):

```

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

```
- etiquetado(instr\_expr(Exp)):

```

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

```
- etiquetado(instr\_if(Exp, Bloq)):

```

    $.prim = etq
    etiquetado(Exp)
    etiquetado-acc-val(Exp)
    etq++
    etiquetado(Bloq)
    $.sig = etq

```

- etiquetado(instr\_if\_else(Exp, Bloq1, Bloq2)):

```
$.prim = etq
etiquetado(Exp)
etiquetado-acc-val(Exp)
etq++
etiquetado(Bloq1)
etq++
$.else = etq
etiquetado(Bloq2)
$.sig = etq
```

- etiquetado(instr\_while(Exp, Bloq)):

```
$.prim = etq
etiquetado(Exp)
etiquetado-acc-val(Exp)
etq++
etiquetado(Bloq)
etq++
$.sig = etq
```

- etiquetado(instr\_read(Exp)):

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

- etiquetado(instr\_write(Exp)):

```
$.prim = etq
etiquetado(Exp)
etq++
$.sig = etq
```

- etiquetado(instr\_nl()):
 

```

$.prim = etq
etq++
$.sig = etq

```
- etiquetado(instr\_new(Exp)):
 

```

$.prim = etq
etiquetado(Exp)
if ref!(Exp.tipo) == tipo_punt
  etq++
end if
etq ++
$.sig = etq

```
- etiquetado(instr\_del(Exp)):
 

```

$.prim = etq
if ref!(Exp.tipo) == tipo_punt
  etq++
end if
etq ++
$.sig = etq

```
- etiquetado(instr\_call(string, ParamsR)):
 

```

$.prim = etq
etq++
etiquetado-paso-param($.vinculo,ParamsR)
etq++
$.sig = etq

```
- etiquetado(instr\_bloque(Bloq)):
 

```

$.prim = etq
etiquetado(Bloq)
$.sig = etq

```

- etiquetado(si\_paramsR(ParamsRL)):
  - \$.prim = etq
  - etiquetado(ParamsRL)
  - \$.sig = etq
- etiquetado(no\_paramsR()):
  - \$.prim = etq
  - \$.sig = etq
- etiquetado(muchos\_paramsR(ParamsRL, Exp)):
  - \$.prim = etq
  - etiquetado(ParamsRL)
  - etiquetado(Exp)
  - \$.sig = etq
- etiquetado(un\_paramsR(Exp)):
  - \$.prim = etq
  - etiquetado(Exp)
  - \$.sig = etq
- etiquetado(asig(Exp1, Exp2)):
  - \$.prim = etq
  - etiquetado(Exp1)
  - etiquetado(Exp2)
  - if ref!(Exp1.tipo) == tipo\_real && ref!(Exp2.tipo) == tipo\_int
  - if es-designador(Exp2)
  - etq += 3
  - else
  - etq += 2
  - end if
  - else
  - etq++
  - end if
  - \$.sig = etq

- etiquetado(menor(Exp1, Exp2)):  
 $\$.prim = etq$   
 $etiquetado-opnds(Exp1, Exp2)$   
 $etq++$   
 $\$.sig = etq$
- etiquetado(mayor(Exp1, Exp2)):  
 $\$.prim = etq$   
 $etiquetado-opnds(Exp1, Exp2)$   
 $etq++$   
 $\$.sig = etq$
- etiquetado(menor\_igual(Exp1, Exp2)):  
 $\$.prim = etq$   
 $etiquetado-opnds(Exp1, Exp2)$   
 $etq++$   
 $\$.sig = etq$
- etiquetado(mayor\_igual(Exp1, Exp2)):  
 $\$.prim = etq$   
 $etiquetado-opnds(Exp1, Exp2)$   
 $etq++$   
 $\$.sig = etq$
- etiquetado(igual(Exp1, Exp2)):  
 $\$.prim = etq$   
 $etiquetado-opnds(Exp1, Exp2)$   
 $etq++$   
 $\$.sig = etq$
- etiquetado(no\_igual(Exp1, Exp2)):  
 $\$.prim = etq$   
 $etiquetado-opnds(Exp1, Exp2)$   
 $etq++$   
 $\$.sig = etq$

- etiquetado(suma(Exp1, Exp2)):
  - \$\$.prim = etq
  - etiquetado-opnds(Exp1, Exp2)
  - etq++
  - \$.sig = etq
- etiquetado(resta(Exp1, Exp2)):
  - \$\$.prim = etq
  - etiquetado-opnds(Exp1, Exp2)
  - etq++
  - \$.sig = etq
- etiquetado(and(Exp1, Exp2)):
  - \$\$.prim = etq
  - etiquetado-opnds(Exp1, Exp2)
  - etq++
  - \$.sig = etq
- etiquetado(or(Exp1, Exp2)):
  - \$\$.prim = etq
  - etiquetado-opnds(Exp1, Exp2)
  - etq++
  - \$.sig = etq
- etiquetado(mult(Exp1, Exp2)):
  - \$\$.prim = etq
  - etiquetado-opnds(Exp1, Exp2)
  - etq++
  - \$.sig = etq
- etiquetado(div(Exp1, Exp2)):
  - \$\$.prim = etq
  - etiquetado-opnds(Exp1, Exp2)
  - etq++
  - \$.sig = etq



- etiquetado(mod(Exp1, Exp2)):
 

```
$.prim = etq
etiquetado-opnds(Exp1, Exp2)
etq++
$.sig = etq
```
- etiquetado(negativo(Exp)):
 

```
$.prim = etq
etiquetado(Exp)
etq-acc-val(Exp)
etq++
$.sig = etq
```
- etiquetado(not(Exp)):
 

```
$.prim = etq
etiquetado(Exp)
etq-acc-val(Exp)
etq++
$.sig = etq
```
- etiquetado(index(Exp1, Exp2)):
 

```
$.prim = etq
etiquetado(Exp1)
etiquetado(Exp2)
etq-acc-val(Exp2)
if ref!(Exp1.tipo) == tipo_array
  etq++
end if
etq += 2
$.sig = etq
```
- etiquetado(acceso(Exp, string)):
 

```
$.prim = etq
```

etiquetado(Exp)

etq += 2

\$.sig = etq

- etiquetado(indireccion(Exp)):

\$.prim = etq

etiquetado(Exp)

etq++

\$.sig = etq

- etiquetado(lit\_ent(string)):

\$.prim = etq

etq++

\$.sig = etq

- etiquetado(lit\_real(string)):

\$.prim = etq

etq++

\$.sig = etq

- etiquetado(true()):

\$.prim = etq

etq++

\$.sig = etq

- etiquetado(false()):

\$.prim = etq

etq++

\$.sig = etq

- etiquetado(lit\_cadena(string)):

\$.prim = etq

etq++

\$.sig = etq

- etiquetado(iden(string)):
 

```

$.prim = etq
if $.vinculo == dec_var
    etiquetado-acc-id($.vinculo)
else if $.vinculo == param
    etiquetado-acc-id($.vinculo)
else if $.vinculo == param_ref
    etiquetado-acc-id($.vinculo)
end if
$.sig = etq

```
- etiquetado(null()):
 

```

$.prim = etq
etq++
$.sig = etq

```
- etiquetado-opnds(Opnd0,Opnd1):
 

```

if ref!(Opnd0.tipo) == tipo_int && ref!(Opnd1.tipo) == tipo_real
    etiquetado(Opnd0)
    etiquetado-acc-val(Opnd0)
    etq++
    etiquetado(Opnd1)
    etiquetado-acc-val(Opnd1)
else if ref!(Opnd0.tipo) == tipo_real && ref!(Opnd1.tipo) ==
tipo_int
    etiquetado(Opnd0)
    etiquetado-acc-val(Opnd0)
    etiquetado(Opnd1)
    etiquetado-acc-val(Opnd1)
    etq++
else
    etiquetado(Opnd0)

```

- ```

    etiquetado-acc-val(Opnd0)
    etiquetado(Opnd1)
    etiquetado-acc-val(Opnd1)
end if

```
- etiquetado-acc-id(dec\_var(T, string)):

```

    if $.nivel = 0 then
        etq++
    else
        etiquetado-acc-var($.vinculo)
    end if

```
  - etiquetado-acc-id(param(T, string)):

```

    etiquetado-acc-var($.vinculo)

```
  - etiquetado-acc-id(param\_ref(T, string)):

```

    etiquetado-acc-var($.vinculo)
    etq++

```
  - etiquetado-acc-var(V):

```

    etq += 3

```
  - etiquetado-acc-val(E):

```

    if es-designador(E) then
        etq++
    end if

```
  - etiquetado-paso-param(dec\_proc(string,ParamsF,Bloq),ParamsR):

```

    etiquetado-paso-param-aux(ParamsF, ParamsR)

```
  - etiquetado-paso-param-aux(si\_paramF(ParamsFL), si\_paramsR(ParamsRL)):

```

    etiquetado-paso-param-aux(ParamsFL, ParamsRL)

```
  - etiquetado-paso-param-aux(no\_paramFL(), no\_paramsRL()):

```

    noop

```

- etiquetado-paso-param-aux(muchos\_paramsF(ParamsFL, Param), muchos\_paramsR(ParamsRL, Exp)):
 

```

      etiquetado-paso-param-aux(ParamsFL, ParamsRL)
      etq =+ 3
      etiquetado(Exp)
      if Param = param_ref(T,id) then
        etq++
      else
        if !es-designador(Exp) then
          if ref!(Exp.tipo) == tipo_int && ref!(Param.tipo) == tipo_real
        then
          etq++
        end if
        etq++
      else
        if ref!(Exp.tipo) == tipo_int && ref!(Param.tipo) == tipo_real
      then
        etq += 3
      else
        etq++
      end if
    end if
  end if

```
- etiquetado-paso-param-aux(un\_paramF(Param), un\_paramsR(Exp)):
 

```

      etq =+ 3
      etiquetado(Exp)
      if Param = param_ref(T,id) then
        etq++
      else
        if !es-designador(Exp) then

```

```

        if ref!(Exp.tipo) == tipo_int && ref!(Param.tipo) == tipo_real
    then
        etq++
    end if
    etq++
else
    if ref!(Exp.tipo) == tipo_int && ref!(Param.tipo) == tipo_real
then
        etq += 3
    end if
    etq++
end if
end if

• recolecta_subs(si_decs(DecsAux)):
    recolecta_subs(DecsAux)

• recolecta_subs(no_decs()):
    noop

• recolecta_subs(muchas_decs(DecsAux, Dec)):
    recolecta_subs(DecsAux)
    recolecta_subs(Dec)

• recolecta_subs(una_dec(Dec)):
    recolecta_subs(Dec)

• recolecta_subs(dec_var(T, string)):
    noop

• recolecta_subs(dec_tipo(T, string)):
    noop

• recolecta_subs(dec_proc(string, ParamsF, Bloq)):
    apila(sub_pendientes, $)

```