Fase 4: Finalización del procesador para Tiny

G01
Esther Babon Arcauz
Pablo Campo Gómez
Claudia López-Mingo Moreno
José Antonio Ruiz Heredia

1. Especificación del proceso de vinculación

1.1. Programa

```
vincula(prog(Bloq)):
        ts = creaTS()
        vincula(Bloq)
```

1.2. Declaraciones

```
vincula(Bloq(DecsOpt, InsOpt)):
        abreAmbito(ts)
        recolectaDecs1(DecsOpt)
        recolectaDecs2(DecsOpt)
        vincula1(InsOpt)
        cierraAmbito(ts)

recolectaDecs1(si_decs(LDecs)):
        recolectaDecs1(LDecs)

recolectaDecs2(si_decs(LDecs)):
        recolectaDecs2(LDecs)

recolectaDecs1(no_decs()):
        noop

recolectaDecs2(no_decs()):
        noop

recolectaDecs1(muchas_decs(LDecs, Dec)):
        recolectaDecs1(LDecs)
        recolectaDec1(Dec)

recolectaDecs2(muchas_decs(LDecs, Dec)):
        recolectaDecs2(LDecs)
        recolectaDec2(Dec)

recolectaDecs1(una_dec(Dec)):
        recolectaDec1(Dec)

recolectaDecs2(una_dec(Dec)):
        recolectaDec2(Dec)

recolectaDec1(dec_var(T, id)):
        vincula1(T)
        if contiene(ts,Id) then
                error
        else
                inserta(ts,Id,$)
        end if

recolectaDec2(dec_var(T, id)):
        vincula2(T)
```

```
recolectaDec1(dec_tipo(T, id)):
        vincula1(T)
        if contiene(ts,Id) then
                error
        else
                inserta(ts,Id,$)
        end if

recolectaDec2(dec_tipo(T, id)):
        vincula2(T)


recolectaDec1(dec_proc(id, PFormOpt, Bloq)):
        if contiene(ts,Id) then
                error
        else
                inserta(ts,Id,$)
        end if
        abreAmbito(ts)
        recolectaDecs1(PformOpt)
        recolectaDecs2(PformOpt)
        vincula(bloq)
        cierraAmbito(ts)

recolectaDec2(dec_proc(id, PFormOpt, Bloq)):
        noop

recolectaDecs1(si_pform(LPForm)):
        recolectaDecs1(LPForm)

recolectaDecs2(si_pform(LPForm)):
        recolectaDecs2(LPForm)

recolectaDecs1(no_pform()):
        noop

recolectaDecs2(no_pform()):
        noop

recolectaDecs1(muchas_pforms(LPForm, PForm)):
        recolectaDecs1(LPForm)
        recolectaDec1(PForm)

recolectaDecs2(muchas_pforms(LPForm, PForm)):
        recolectaDecs2(LPForm)
        recolectaDec2(PForm)

recolectaDecs1(una_pform(PForm)):
        recolectaDec(PForm)

recolectaDecs2(una_pform(PForm)):
        recolectaDec2(PForm)
```

```
recolectaDec1(pform_ref(T,id)):
        vincula1(T)
        if contiene(ts,Id) then
                error
        else
                inserta(ts,id,$)
        end if

recolectaDec2(pform_ref(T,id)):
        vincula2(T)

recolectaDec1(pform_no_ref(T,id)):
        vincula1(T)
        if contiene(ts,Id) then
                error
        else
                inserta(ts,id,$)
        end if

recolectaDec2(pform_no_ref(T,id)):
        vincula2(T)
```

1.3. Tipos

```
vincula1(array(T, dim)):
        vincula1(T)

vincula2(array(T, dim)):
        vincula2(T)

vincula1(puntero(T)):
        if T != iden(_):
                vincula1(T)
        end if

vincula2(puntero(T)):
        if (T =iden(Id)):
                T.vinculo = vinculoDe(ts, Id)
                 if (T.vinculo = -|):
                          error
                end if
        else
                 vincula2(T)
        end if

vincula1(iden(id)):
        if contiene(ts,id) then
                $.vinculo = vinculoDe(ts,id)
        else
                error
        end if

vincula2(iden(id)):
        noop
```

```
vincula1(struct(LCamp)):
        vincula1(LCamp)

vincula2(struct(LCamp)):
        vincula2(LCamp)

vincula1(lit_ent(string)):
        noop

vincula2(lit_ent(string)):
        noop

vincula1(lit_real(string)):
        noop

vincula2(lit_real(string)):
        noop

vincula1(lit_bool(string)):
        noop

vincula2(lit_bool(string)):
        noop

vincula1(lit_string(string)):
        noop

vincula2(lit_string(string)):
        noop

vincula1(muchos_camp(LCamp,  Camp)):
        vincula1(LCamp)
        vincula1(Camp)

vincula2(muchos_camp(LCamp,  Camp)):
        vincula2(LCamp)
        vincula2(Camp)

vincula1(un_camp(Camp)):
        vincula1(Camp)

vincula2(un_camp(Camp)):
        vincula2(Camp)

vincula1(camp(T, Id)):
        vincula1(T)

vincula2(camp(T, Id)):
        vincula2(T)
```

1.4. Instrucciones

```
vincula1(Si_ins(LIns)):
        vincula1(LIns)

vincula1(no_ins()):
        noop

vincula1(muchas_ins(LIns, Ins)):
        vincula1(LIns)
        vincula1(Ins)

vincula1(una_ins(Ins)):
        vincula1(Ins)

vincula1(ins_asig(Exp)):
        vincula1(Exp)

vincula1(ins_if(Exp, Bloq)):
        vincula1(Exp)
        vincula(Bloq)

vincula1(ins_if_else(Exp, Bloq, Bloq)):
        vincula1(Exp)
        vincula(Bloq)
        vincula(Bloq)

vincula1(ins_while(Exp, Bloq)):
        vincula1(Exp)
        vincula(Bloq)

vincula1(ins_read(Exp)):
        vincula1(Exp)

vincula1(ins_write(Exp)):
        vincula1(Exp)

vincula1(ins_nl()):
        noop

vincula1(ins_new(Exp)):
        vincula1(Exp)

vincula1(ins_delete(Exp)):
        vincula1(Exp)
```

```
vincula1(ins_call(Iden, PRealOpt)):
        if contiene(ts,Iden) then
                $.vinculo = vinculoDe(ts,Iden)
        else
                error
        end if
        vincula1(PRealOpt)

vincula1(ins_bloque(Bloq)):
        vincula(Bloq)
```

1.5. Expresiones

```
vincula1(si_preal(LPReal)):
        vincula1(LPreal)

vincula1(no_preal()):
        noop

vincula1(muchos_preal(LPReal, Exp)):
        vincula1(LPReal)
        vincula1(Exp)

vincula1(un_preal(Exp)):
        vincula1(Exp)

vincula1(asig(Exp0, Exp1)):
        vincula1(Exp0)
        vincula1(Exp1)

vincula1(mayor(Exp0, Exp1)):
        vincula1(Exp0)
        vincula1(Exp1)

vincula1(menor(Exp0, Exp1)):
        vincula1(Exp0)
        vincula1(Exp1)

vincula1(mayorIgual(Exp0, Exp1)):
        vincula1(Exp0)
        vincula1(Exp1)

vincula1(menorIgual(Exp0, Exp1)):
        vincula1(Exp0)
        vincula1(Exp1)

vincula1(igual(Exp0, Exp1)):
        vincula1(Exp0)
        vincula1(Exp1)
```

```
vincula1(desigual(Exp0, Exp1)):
        vincula1(Exp0)
        vincula1(Exp1)

vincula1(suma(Exp0, Exp1)):
        vincula1(Exp0)
        vincula1(Exp1)

vincula1(resta(Exp0, Exp1)):
        vincula1(Exp0)
        vincula1(Exp1)

vincula1(and(Exp0, Exp1)):
        vincula1(Exp0)
        vincula1(Exp1)

vincula1(or(Exp0, Exp1)):
        vincula1(Exp0)
        vincula1(Exp1)

vincula1(mul(Exp0, Exp1)):
        vincula1(Exp0)
        vincula1(Exp1)

vincula1(div(Exp0, Exp1)):
        vincula1(Exp0)
        vincula1(Exp1)

vincula1(mod(Exp0, Exp1)):
        vincula1(Exp0)
        vincula1(Exp1)

vincula1(neg(Exp)):
        vincula1(Exp)

vincula1(not(Exp)):
        vincula1(Exp)

vincula1(acceso_array(Exp0, Exp1)):
        vincula1(Exp0)
        vincula1(Exp1)

vincula1(acceso_campo(Exp, id)):
        vincula1(Exp)

vincula1(acceso_puntero(Exp)):
        vincula1(Exp)
```

```
vincula1(exp_litEntero(N)):
        noop

vincula1(exp_litReal(R)):
        noop

vincula1(exp_litCadena(Id)):
        noop

vincula1(exp_Identificador(id)):
        if contiene(ts,id) then
                $.vinculo = vinculoDe(ts,id)
        else
                error
        end if

vincula1(exp_litBoolTrue()):
        noop

vincula1(exp_litBoolFalse()):
        noop

vincula1(exp_null()):
        noop
```

2. Especificación del procedimiento de comprobación de tipos.

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

2.1. Comprobación de tipos de Programa.

```
tipado(prog(Bloq) =
        tipado(Bloq)
        $.tipo = Bloq.tipo

tipado(bloq(DecsOpt,InsOpt)) =
        tipado(DecsOpt)
        tipado(InsOpt)
        $.tipo = ambos-ok(InsOpt.tipo,DecsOpt.tipo)
```

2.2. Comprobación de tipos de declaraciones.

```
tipado(si_decs(LDecs)) =
        tipado(LDecs)
        $.tipo = LDecs.tipo

tipado(no_decs()) =
        $.tipo = OK

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

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

tipado(dec_var(T, Iden)) =
        tipado(T)
        $.tipo = T.tipo

tipado(dec_tipo(T, Iden)) =
        tipado(T)
        $.tipo = T.tipo

tipado(dec_proc(Iden,PFormOpt, Bloq)) =
        tipado(PFormOpt)
        tipado(Bloq)
        $.tipo = ambos-ok(PFormOpt.tipo, Bloq.tipo)

tipado(si_pform(LPForm)) =
        tipado(LPForm)
        $.tipo = LPForm.tipo

tipado(no_pform()) =
        $.tipo = OK

tipado(una_pform(PForm)) =
        tipado(PForm)
        $.tipo = PForm.tipo
```

```
tipado(muchas_PForms(LPForm, PForm)) =
        tipado(LPForm)
        tipado(PForm)
        $.tipo = ambos-ok(LPForm.tipo, PForm.tipo)

tipado(pform_no_ref(T, Iden)) =
        tipado(T)
        $.tipo = T.tipo
tipado(pform_ref(T, Iden)) =
        tipado(T)
        $.tipo = T.tipo
```

2.3. Comprobación de tipos de tipos básicos.

```
tipado(lit_ent()) =
        $.tipo = ok

tipado(lit_real()) =
        $.tipo = ok

tipado(lit_bool()) =
        $.tipo = ok

tipado(lit_string()) =
        $.tipo = ok

tipado(array(T, litEntero)) =
        si litEntero < 0:
                $.tipo = error
        sino:
                tipado(T)
                $.tipo = T.tipo

tipado(puntero(T)) =
        tipado(T)
        $.tipo = T.tipo

tipado(struct(LCamp)) =
        si !hayRepetidos(LCamp):
                tipado(LCamp)
                $.tipo = LCamp.tipo
        sino:
                $.tipo = error
```

```
hayRepetidos(LCamp):
        for(int i = 0; i < LCamp.length; i++):
                for(int j = 0; j < LCamp.length; j++):
                si LCamp[i] == LCamp[j] && i != j:
                        return true
        return false

tipado(un_campo(Camp)) =
        tipado(Camp)
        $.tipo = C.tipo

tipado(muchos_campos(LCamp, Camp)) =
        tipado(LCamp)
        tipado(Camp)
        $.tipo = ambos-ok(LCamp.tipo, Camp.tipo)

tipado(campo(id,T)) =
        tipado(T)
        $.tipo = T.tipo

tipado(iden(string)) =
        si $.vinculo = dec_tipo(T, id)añadir
                tipado(T)
                $.tipo = T.tipo

        si no:
                $.tipo = error
```

2.4. Comprobación de tipos de instrucciones.

```
global st <- vacio()
```

//Explicación: Si ya se ha hecho esta comprobación antes se devuelve true directamente. Si no se ha hecho se añaden y se mira si son compatibles (string solo con otro string, bool solo con otro bool etc). Si lo son se deja metido y sino se saca

```
son_compatibles(T0,T1) =
        si estan(st, (T0, T1)):
                return true
        si no:
                añadir(st, (T0, T1))
        si ref!(T0) == int && ref!(T1) == int:
                return true
        si ref!(T0) == real && (ref!(T1) == int || ref!(T1) == real):
                return true
        si ref!(T0) == bool && ref!(T1) == bool:
                return true
        si ref!(T0) == string && ref!(T1) == string:
                return true
        si ref!(T0) == array(Iden0, T2) && ref!(T1) == array(Iden1, T3):
                si n0 == n1 && son_compatibles(T2, T3):
                        return true
        si ref!(T0) == struct(lc1) && ref!(t1) == struct(lc2):
                si campos_compatibles(lc1, lc2):
                        return true
        si ref!(T0) == puntero(T2) && T1 == null:
                return true
        si ref!(T0) == puntero(T2) && ref!(T1) == puntero(T3):
                si son_compatibles(T2, T3):
                        return true
        eliminar(st, (T0, T1))
        return false

//Explicación: Dos campos solo son compatibles si son
        1.un_camp y tipos compatibles
        2.muchos_camp y tanto sus LCamp como los tipos lo son

campos_compatibles(un_campo(campo(Iden1, T1)),
un_campo(campo(Iden2, T2))) :
        return son_compatibles(T1, T2)

campos_compatibles(un_campo(campo(Iden1, T1)),
muchos_campos(LCamp, campo(Iden2, T2))) :
        return false

campos_compatibles(muchos_camp(LCamp, camp(Iden1, T1)),
un_camp(camp(Iden2, T2)) :
        return false
```

```
campos_compatibles(muchos_camp(LCamp1,camp(Iden1,T1)),
muchos_camp(LCamp2, camp(Iden2,T2)) :
        return ambos-ok(campos_compatibles(LCamp1,LCamp2),
son_compatibles(T1,T2))

tipado(si_ins(LIns)) :
        Tipado(LIns)
        $.tipo = LIns.tipo

tipado(no_ins()) :
        $.tipo = OK

tipado(una_ins(Ins)):
        tipado(Ins)
        $.tipo = Ins.tipo

tipado(muchas_ins(LIns, Ins)) :
        tipado(LIns)
        tipado(Ins)
        $.tipo = ambos-ok(LIns.tipo, Ins.tipo)

tipado(ins_asig(Exp)) :
        si tipado(Exp) == ok:
                $.tipo = ok
        si no:
                $.tipo = ERROR

tipado(ins_if(Exp, Bloq)):
        tipado(Exp)
        si ref!(Exp.tipo) == bool:
                si tipado(Bloq) == ok:
                        $.tipo = ok
                si no:
                        $.tipo = ERROR

        si no:
                $.tipo = ERROR

tipado(ins_if_else(Exp,Bloq1, Bloq2)) :
        tipado(Exp)
        si ref!(E.tipo) == bool
                si tipado(Bloq1) == OK && tipado(Bloq2) == OK:
                        $.tipo = OK
                si no:
                        $.tipo = ERROR
        si no:
                $.tipo = ERROR
```

```
tipado(ins_while(Exp, Bloq)) :
        tipado(Exp)
        si ref!(Exp.tipo) == bool:
                si tipado(Bloq) == ok:
                        $.tipo = ok
                si no:
                        $.tipo = ERROR

        si no:
                $.tipo = ERROR

tipado(ins_read(Exp)) :
        tipado(Exp)
        si (ref!(Exp.tipo) == int || ref!(Exp.tipo) == real || ref!(Exp.tipo) == string)
&&      es_desig(Exp):
                $.tipo = OK
        si no:
                $.tipo = ERROR

es_desig(Exp):
        si E == iden || E == acceso_array || E == acceso_puntero || E == acceso_struct:
                return true
        return false

tipado(ins_write(Exp)) :
        tipado(Exp)
        si (ref!(Exp.tipo) == int || ref!(Exp.tipo) == real || ref!(Exp.tipo) == bool ||
ref!(Exp.tipo) == string):
                $.tipo = OK
        si no:
                $.tipo = ERROR

tipado(ins_nl):
        $.tipo = OK

tipado(ins_new(Exp)) :
        tipado(Exp)
        si ref!(Exp.tipo) == puntero:
                $.tipo = OK
        si no:
                $.tipo = ERROR

tipado(ins_delete(Exp)) :
        tipado(Exp)
        si ref!(Exp.tipo) == puntero:
                $.tipo = OK
        si no:
                $.tipo = ERROR

tipado(ins_bloque(Bloq)) :
```

```
        tipado(Bloq)
        $.tipo = Bloq.tipo

tipado(ins_call(string PRealOpt)):
        tipado(PRealOpt)
        let string.vinculo = dec_proc  dec_proc = dec_proc(id,PForm,_)
        PRealOpt.tipo = chequeo_params(PForm, PRealOpt)
        si PRealOpt.tipo = ok:
                $.tipo = ok
        si no:
                $.tipo = error

chequeo_params(no_PForm, no_PReal): return ok

chequeo_params(un_PForm, un_PReal):
        return chequeo_params(E, PForm)

chequeo_params(Muchos_PReal(LPReal, Exp),
muchos_PForm(LPForm,PForm)):

        return ambos_ok(chequeo_params(LPReal,LPForm),
        chequeo_parametro(Exp,PForm))

chequeo_params(Exp, PForm_no_ref(id, T)):
        tipado(Exp)
        tipado(PForm_no_ref)
        si son_compatibles(T, E.tipo):
                return ok
        si no:
                return error

chequeo_params(Exp, PForm_ref(id,T)) =
        tipado(Exp)
        tipado(PForm_ref)
        si es_desig(Exp) && son_compatibles(T, Exp.tipo)
                return ok
        si no:
                return error
```

2.5. Comprobación de tipos de expresiones.

```
tipado(si_preal(LPReal)):
        tipado(LPReal)
        $.tipo = LPReal.tipo

tipado(no_preal()) :
        $.tipo = OK

tipado(un_preal(PReal)):
```

```
        tipado(PReal)
        $.tipo = PReal.tipo

tipado(muchos_PReal(LPReal, Exp)) :
        tipado(LPReal)
        tipado(Exp)
        $.tipo = ambos-ok(LPReal.tipo, Exp.tipo)

tipado(exp_litEntero(N)) :
        $.tipo = int

tipado(exp_litReal(R)) :
        $.tipo = real

tipado(exp_litEntero(N)):
        $.tipo = int

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

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

tipado(exp_litCadena(String)) :
        $.tipo = string

tipado(exp_identificador(String)) :
        si $.vinculo == dec_var(String, T) || $.vinculo == pform_ref(T, String) ||
$.vinculo == pform_no_ref(T, String):
                $.tipo = T
        si no:
                $.tipo = ERROR

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

tipado(menor(Exp0, Exp1)) :
        tipado(Exp0)
        tipado(Exp1)
        $.tipo = tipo_relacional(E0.tipo, E1.tipo)



tipado(menorIgual(Exp0, Exp1)):
```

```
        tipado(Exp0)
        tipado(Exp1)
        $.tipo = tipo_relacional(Exp0.tipo, Exp1.tipo)

tipado(mayor(Exp0, Exp1)):
        tipado(Exp0)
        tipado(Exp1)
        $.tipo = tipo_relacional(Exp0.tipo, Exp1.tipo)

tipado(MayorIgual(Exp0, Exp1)):
        tipado(Exp0)
        tipado(Exp1)
        $.tipo = tipo_relacional(Exp0.tipo, Exp1.tipo)

tipo_relacional(T0, T1):
        si (ref!(T0) == int || ref!(T0) == real) && (ref!(T1) == int || ref!(T1) == real):
                return bool
        si no:
                si ref!(T0) == bool && ref!(T1) == bool:
                        return bool
                si no:
                si ref!(T0) == string && ref!(T1) == string:
                        return bool
                si no:
                        return error

tipado(igual(Exp0, Exp1)) :
        tipado(Exp0)
        tipado(Exp1)
        $.tipo = tipo_relacional2(Exp0.tipo, Exp1.tipo)

tipado(desigual(Exp0, Exp1)) :
        tipado(Exp0)
        tipado(Exp1)
        $.tipo = tipo_relacional2(Exp0.tipo, Exp1.tipo)




tipo_relacional2(T0, T1):
        si (ref!(T0) == int || ref!(T0) == real) && (ref!(T1) == int || ref!(T1) == real):
```

```
                return bool
        si no:
                si ref!(T0) == bool && ref!(T1) == bool:
                        return bool
                si no:
                        si ref!(T0) == string && ref!(T1) == string:
                                return bool
        si ref!(T0) == puntero && ref!(T1) == puntero:
                return bool
        si no:
                si (ref!(T0) == puntero && ref!(T1) == null) || (ref!(T0) == null &&
                         ref!(T1) == puntero):
                return bool
        si no:
                si ref!(T0) == null && ref!(T1) == null:
                        return bool
                si no:
                        return error

tipado(and(Exp0, Exp1)) :
        tipado(Exp0)
        tipado(Exp1)
        $.tipo = tipo_logico(Exp0.tipo, Exp1.tipo)

tipado(or(Exp0, Exp1)) :
        tipado(Exp0)
        tipado(Exp1)
        $.tipo = tipo_logico(Exp0.tipo, Exp1.tipo)

tipo_logico(T0, T1) :
        si ref!(T0) == bool && ref!(T1) == bool:
                return bool
        si no:
                return error

tipado(suma(Exp0, Exp1)) :
        tipado(Exp0)
        tipado(Exp1)
        $.tipo = tipo_binat(Exp0.tipo, Exp1.tipo)


tipado(resta(Exp0, Exp1)) :
        tipado(Exp0)
        tipado(Exp1)
```

```
        $.tipo = tipo_binat(Exp0.tipo, Exp1.tipo)

tipado(mul(Exp0, Exp1)) :
        tipado(Exp0)
        tipado(Exp1)
        $.tipo = tipo_binat(Exp0.tipo, Exp1.tipo)

tipado(div(Exp0, Exp1)) :
        tipado(Exp0)
        tipado(Exp1)
        $.tipo = tipo_binat(Exp0.tipo, Exp1.tipo)

tipo_binat(T0, T1) :
        si ref!(T0) == int && ref!(T1) == int:
                return int
        si no:
                si (ref!(T0) == real && (ref!(T1) == int || ref!(T1) == real)) || (ref!(T1)
== real && (ref!(T0) == int || ref!(T0) == real)):
                        return real
                si no:
                        return error

tipado(mod(Exp0, Exp1)) :
        tipado(Exp0)
        tipado(Exp1)
        tipo_mod_ent(Exp0.tipo, Exp1.tipo)
        si ref!(T0) == int && ref!(T1) == int:
                        $.tipo = int
                si no:
                        $.tipo = error

tipado(neg(Exp)) :
        tipado(Exp)
        si ref!(Exp.tipo) == int:
                $.tipo = int
        si no si ref!(Exp.tipo) == real:
                $.tipo = real
        si no:
                $.tipo = ERROR


tipado(not(Exp)) :
        tipado(Exp)
        si ref!(Exp.tipo) == bool:
```

```
                $.tipo = bool
        si no:
                $.tipo = ERROR

tipado(acceso_array(Exp0, Exp1)) :
        tipado(Exp0)
        tipado(Exp1)
        si ref!(Exp0.tipo) = array(n, tb)
                si ref!(Exp1.tipo) == int
                        $.tipo = tb
                si no
                        return error
        si no
                return error

tipado(acceso_campo(Exp, c)) :
        tipado(Exp)
        si ref!(Exp.tipo) == struct(LCampos):
                        $.tipo = tipo_de(LCampos, c)
        si no:
                $.tipo = error

tipo_de(LCampos, c):
        i = 0
        mientras i < LCampos.length:
                        sea LCampos[i] = campo(id, T):
                                si c == id:
                                        return T
        return error

tipado(acceso_puntero(Exp)):
        tipado(Exp)
        si ref!(Exp.tipo) == puntero(T):
                $.tipo = T
        si no:
                $.tipo = error




tipado(asig(Exp0,Exp1)):
        tipado(Exp0)
        tipado(Exp1)
```

```
        if es_desig(Exp0) then
                if compatibles(Exp0.tipo, Exp1.tipo) then
                        $.tipo = ok
                else:
                        $.tipo = error
                end if
        else:
                 $.tipo = error
        end if
```

3. Especificación del procesamiento de asignación de espacio.

```
var dir = 0
var nivel = 0
var max_dir = 0
var desplazamiento = 0

asig_espacio(prog(Bloque)):
        asig_espacio(Bloque)

asig_espacio(bloque(DecsOpt, InsOpt)):
        dir_ant = dir
        asig_espacio(DecsOpt)
        asig_espacio(InsOpt)
        dir = dir_ant

asig_espacio(si_decs(LDecs)):
        asig_espacio1(LDecs)
        asig_espacio2(LDecs)

asig_espacio(no_decs()):
        skip
```

3.1. Primera pasada en la seccion de declaraciones

```
asig_espacio1(muchas_decs(LDecs, Dec)):
        asig_espacio1(LDecs)
        asig_espacio1(Dec)

asig_espacio1(una_dec(Dec)):
        asig_espacio1(Dec)



asig_espacio1(dec_var(Tipo, Iden)):
        $.dir = dir
        $.nivel = nivel
```

```
        asig_tam1(Tipo)
        dir = dir + Tipo.tam

asig_espacio1(dec_tipo(Tipo, Iden)):
        asig_tam1(Tipo)

asig_espacio1(dec_proc(Iden, PFormOpt, Bloque)):
        dir_ant = dir
        max_dir_ant = max_dir
        nivel++
        $.nivel = nivel
        dir = 0
        max_dir = 0
        asig_espacio1(PFormOpt)
        asig_espacio2(PFormOpt)
        asig_espacio(Bloque)
        $.tam = dir
        dir = dir_ant
        max_dir = max_dir_ant
        nivel = nivel - 1

asig_espacio1(si_pform(LPForm)):
        asig_espacio1(LPForm)

asig_espacio1(no_pform()):
        skip

asig_espacio1(muchas_pforms(LPForm, PForm)):
        asig_espacio1(LPForm)
        asig_espacio1(PForm)

asig_espacio1(una_pform(PForm)):
        asig_espacio1(PForm)

asig_espacio1(pref(Tipo, Iden)):
        $.dir = dir
        $.nivel = nivel
        asig_tam1(Tipo)
        dir = dir + 1

asig_espacio1(pnoref(Tipo,Iden)):
        $.dir = dir
        $.nivel = nivel
        asig_tam1(Tipo)
        dir = dir + Tipo.tam
```

3.2. Segunda pasada en la sección de declaraciones

```
asig_espacio2(muchas_decs(LDecs, Dec)):
        asig_espacio2(LDecs)
```

```
        asig_espacio2(Dec)

asig_espacio2(una_dec(Dec)):
        asig_espacio2(Dec)

asig_espacio2(dec_var(Tipo, Iden)):
        asig_tam2(Tipo)

asig_espacio2(dec_tipo(Tipo, Iden)):
        asig_tam2(Tipo)

asig_espacio2(dec_proc(Iden, PFormOpt, Bloque)):
        skip

asig_espacio2(si_pform(LPForm)):
        asig_espacio2(LPForm)

asig_espacio2(no_pform()):
        skip

asig_espacio2(muchas_pforms(LPForm, PForm)):
        asig_espacio2(LPForm)
        asig_espacio2(PForm)

asig_espacio2(una_pform(PForm)):
        asig_espacio2(PForm)

asig_espacio1(pref(Tipo, Iden)):
        $.dir = dir
        $.nivel = nivel
        asig_tam1(Tipo)
        dir = dir + 1

asig_espacio1(pnoref(Tipo,Iden)):
        $.dir = dir
        $.nivel = nivel
        asig_tam1(Tipo)
        dir = dir + Tipo.tam
```

3.3. Asignación del tamaño de los tipos.

```
asig_tam1(int()):
        $.tam = 1
```

```
asig_tam2(int()):
        skip

asig_tam1(real()):
        $.tam = 1

asig_tam2(real()) :
        skip

asig_tam1(bool()):
        $.tam = 1

asig_tam2(bool()):
        skip

asig_tam1(string()):
        $.tam = 1

asig_tam2(string()):
        skip

asig_tam1(puntero(Tipo)):
        if Tipo != iden(string)
                asig_tam1(Tipo)
        $.tam = 1

asig_tam2(puntero(Tipo)):
        if Tipo = iden(string)
                sea Tipo.vinculo = dec_tipo(T', id) :
                        Tipo.tam = T'.tam
        else
                asig_tam2(Tipo)

asig_tam1(iden(string)):
        sea $.vinculo = dec_tipo(T, id):
                $.tam = T.tam

asig_tam2(iden(string)):
        skip

asig_tam1(struct(LCampos)):
        $.tam = asig_tam1(LCampos)

asig_tam2(struct(LCampos)):
        asig_tam2(LCampos)

asig_tam1(un_campo(Campo)):
```

```
        sea Campo = campo(T, id):
                asig_tam1(T)
                Campo.desp = desplazamiento
                return desplazamiento + T.tam

asig_tam2(un_campo(Campo)):
        sea Campo = campo(T, id):
                Campo.desp = desplazamiento
                asig_tam2(T)

asig_tam1(muchos_campos(LCampos, Campo)):
        d_act = asig_tam1(LCampos)
        sea Campo = campo(T, id):
                asig_tam1(T)
                Campo.desp = d_act
                return d_act + T.tam

asig_tam2(muchos_campos(LCampos, Campo)) =
        asig_tam2(LCampos)
        sea Campo = campo(T, id):
                asigna_tam2(T)

asig_tam1(array(Tipo, n)):
        asig_tam1(Tipo)
        $.tam = val(n) * Tipo.tam

asig_tam2(array(Tipo, n)):
        asig_tam2(Tipo)
```

3.4. Asignación del espacio de instrucciones.

```
asig_espacio(si_ins(LIns)):
        asig_espacio(LIns)

asig_espacio(no_ins()):
        skip

asig_espacio(una_ins(Ins)):
        asig_espacio(Ins)

asig_espacio(muchas_ins(LIns, Ins)):
        asig_espacio(LIns)
        asig_espacio(Ins)

asig_espacio(ins_asig(E)):
        skip

asig_espacio(ins_if(E, Bloque)):
```

```
        asig_espacio(Bloque)

asig_espacio(ins_if_else(E, Bloque1, Bloque2)):
        asig_espacio(Bloque1)
        asig_espacio(Bloque2)
asig_espacio(ins_while(E, Bloque)):
        asig_espacio(Bloque)

asig_espacio(ins_read(E)):
        skip
asig_espacio(ins_write(E)):
        skip
asig_espacio(ins_nl()):
        skip
asig_espacio(ins_new(E)):
        skip
asig_espacio(ins_delete(E)):
        skip
asig_espacio(ins_bloque(Bloque)):
        asig_espacio(Bloque)

asig_espacio(ins_call(E)):
        skip
```

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

Instrucciones que hacen falta incluir para extender la máquina-p y que así admita todas las operaciones de Tiny:

apila_int(I): Apila un valor I de tipo int
apila_real(R): Apila un valor R de tipo real
apila_bool(B): Apila un valor B de tipo bool
apila_string(S): Apila un valor S de tipo string

apila_dir(d): Apila el valor que se encuentra en la dirección d de la memoria de datos
desapila_dir(d): Desapila la cima y almacena su valor en la dirección d de la memoria de datos

apila_ind: Desapila un valor d de la cima de la pila y lo usa como dirección de la memoria de datos cuyo valor apila.
desapila_ind: Desapila un valor v de la cima de la pila y un segundo valor d de la subcima. Almacena el valor v en la posición d de la memoria de datos.

apilad(n): Apila el valor del display de nivel n
desapilad(n): Desapila el valor del display de nivel n

suma_int: desapila dos valores enteros y apila el resultado de su suma
resta_int: desapila dos valores enteros y apila el resultado de su resta (subcima - cima)
mul_int: desapila dos valores enteros y apila el resultado de su multiplicación
div_int: desapila dos valores enteros y apila el resultado de su división(subcima/cima)
neg_int: desapila un valor entero y apila el resultado de su negación

suma_real: desapila dos valores reales y apila el resultado de su suma
resta_real: desapila dos valores reales y apila el resultado de su resta (subcima - cima)
mul_real: desapila dos valores reales y apila el resultado de su multiplicación
div_real: desapila dos valores reales y apila el resultado de su división(subcima/cima)
neg_real: desapila un valor real y apila el resultado de su negación

mod: desapila dos valores y apila el resultado de “subcima % cima”

not: desapila un valor y apila el resultado de su negación lógica
and: desapila dos valores y apila el resultado de su conjunción lógica
or: desapila dos valores y apila el resultado de su disyunción lógica

menor_int: desapila dos valores enteros y apila el resultado de “subcima < cima”
menor_real: desapila dos valores reales y apila el resultado de “subcima < cima”
menor_bool: desapila dos valores booleanos y apila el resultado de “subcima < cima”
menor_string: desapila dos cadenas y apila el resultado de “subcima < cima”

mayor_int: desapila dos valores enteros y apila el resultado de “subcima > cima”
mayor_real: desapila dos valores reales y apila el resultado de “subcima > cima”
mayor_bool: desapila dos valores booleanos y apila el resultado de “subcima > cima”
mayor_string: desapila dos cadenas y apila el resultado de “subcima > cima”

menor_igual_int: desapila dos valores enteros y apila el resultado de “subcima <= cima”
menor_igual_real: desapila dos valores reales y apila el resultado de “subcima <= cima”
menor_igual_bool: desapila dos valores booleanos y apila el resultado de “subcima <= cima”
menor_igual_string: desapila dos cadenas y apila el resultado de “subcima <= cima”

mayor_igual_int: desapila dos valores enteros y apila el resultado de “subcima >= cima”
mayor_igual_real: desapila dos valores reales y apila el resultado de “subcima >= cima”
mayor_igual_bool: desapila dos valores booleanos y apila el resultado de “subcima >= cima”
mayor_igual_string: desapila dos cadenas y apila el resultado de “subcima >= cima”

igual_int: desapila dos valores enteros y apila el resultado de “subcima == cima”
igual_real: desapila dos valores reales y apila el resultado de “subcima == cima”
igual_bool: desapila dos valores booleanos y apila el resultado de “subcima == cima”
igual_string: desapila dos cadenas y apila el resultado de “subcima == cima”
igual_puntero: desapila dos punteros y apila el resultado de “subcima == cima”
igual_null: desapila dos tipos null y apila el resultado de “subcima == cima”
igual: desapila un valor null y un valor puntero y apila el resultado de “subcima == cima”

desigual_int: desapila dos valores enteros y apila el resultado de “subcima != cima”
desigual_real: desapila dos valores reales y apila el resultado de “subcima != cima”
desigual_bool: desapila dos valores booleanos y apila el resultado de “subcima != cima”
desigual_string: desapila dos cadenas y apila el resultado de “subcima != cima”
desigual_puntero: desapila dos punteros y apila el resultado de “subcima != cima”
desigual_null: desapila dos tipos null y apila el resultado de “subcima != cima”
desigual: desapila un valor null y un valor puntero y apila el resultado de “subcima != cima”

mueve(n): copia n celdas desde la dirección de origen, extraída de la cima de la pila, a la dirección destino, de la subcima.

ir_a(d): salta de manera incondicional a la instrucción en la dirección d de memoria de programa
ir_v(d): salta a la instrucción en la dirección d de la memoria de programa si el valor de la cima es true y desapila este valor
ir_f(d): salta a la instrucción en la dirección d de la memoria de programa si el valor de la cima es false y desapila este valor
ir_ind(): desapila un valor d de la cima de la pila y salta a la instrucción en la dirección d de la memoria de programa.

alloc(t): con un tipo t. Se reserva una zona de memoria adecuada para almacenar valores del tipo . La operación en sí devuelve la dirección de comienzo de dicha zona de memoria.

dealloc(d, t): con d una dirección, y un tipo. Se notifica que la zona de memoria que comienza en d y que permite almacenar valores del tipo queda liberada.

fetch(d): on d una dirección. Devuelve el valor almacenado en la celda direccionada por d.

store(d,v): con d una dirección, y v un valor. Almacena v en la celda direccionada por d.

copy(t): Con t el tamaño de un tipo. Desapila 2 direcciones y copia el contenido de tamaño t empezando por d1 en un espacio que empieza en d2.

indx(d,i, t): con d una dirección, i un valor, y un tipo. Considera que, a partir de d, comienza un array cuyos elementos son valores del tipo , y devuelve la dirección de comienzo del elemento i-esimo de dicho array.

acc(d,c, t): con d una dirección, c un nombre de campo, y un tipo record. Considera que, a partir de d, está almacenado un registro de tipo , que contiene un campo c. Devuelve la dirección de comienzo de dicho campo.

activa(n,t,d): Reserva espacio en el segmento de pila de registros de activación para un procedimiento con nivel de anidamiento n y tamaño de datos locales t. Almacena la dirección d en la zona de control del registro como dirección de retorno. Almacena en la zona de control el valor del display de nivel n. Apila la dirección de comienzo de los datos en el registro

desactiva(n,t): Libera el espacio ocupado por el registro de activación actual, restaura el estado de la máquina. n es el nivel de anidamiento del procedimiento; t el tamaño de los datos locales. Apila la dirección de retorno; Restaura el valor del display de nivel n al antiguo valor guardado en el registro; Decrementa el puntero de pila de registros de activación en el tamaño ocupado por el registro

dup(): Duplica la cima de la pila

stop(): Detiene la ejecución de la máquina

int2real(): desapila un valor entero y apila el mismo valor convertido a real

leer_entrada_int(): lee un valor entero de la entrada estandar y lo apila
leer_entrada_real): lee un valor real de la entrada estandar y lo apila
leer_entrada_string(): leerá una cadena de la entrada estandar y lo apila

mostrar_int(): imprime en salida estandar el dato entero de la cima y lo desapila.
mostrar_real(): imprime en salida estandar el dato real de la cima y lo desapila.
mostrar_bool(): imprime en salida estandar el dato booleano de la cima y lo desapila.
mostrar_string(): imprime en salida estandar la cadena de la cima y lo desapila.
nl(): mostrará en la salida estándar un salto de línea.

desechar(): desapila la cima y desecha el valor.

5. Especificación del procesamiento de etiquetado.

5.1. Etiquetado para el programa.

```
global procs = pila_vacia()
global etq = 0

etiqueta(prog(Bloq)):
        etiqueta(Bloq)

etiqueta(Bloq(DecsOpt, InsOpt)):
        $.prim = etq
        etiqueta(DecsOpt)
        etq <- etq + 1
        etiqueta(InsOpt)
        $.sig = etq

etiqueta(si_decs()):
        etiqueta(LDecs)

etiqueta(no_dec()): noop

etiqueta(muchas_decs(LDecs, Dec)):
```

```
        etiqueta(LDecs)
        etiqueta(Dec)

etiqueta(una_dec(Dec)):
        etiqueta(Dec)
```

5.2. Etiquetado para las instrucciones.

```
etiqueta(si_ins(LIns)):
        etiqueta(LIns)

etiqueta(no_ins()):
        noop

etiqueta(una_ins(Ins)) =
        etiqueta(Ins)

etiqueta(muchas_ins(LIns, Ins)) =
        etiqueta(LIns)
        etiqueta(Ins)

etiqueta(ins_asig(Exp)):
        etiqueta(Exp)
        t = ref!(Exp.tipo)
        si t == int || t == real || t == string
                si es_desig(Exp)
                        etq <- etq + 1
                etq <- etq + 2
        sino
                etq <- etq + 1

etiqueta(ins_if(Exp, Bloq)):
        $.prim = etq
        etiqueta(Exp)
        si es_desig(Exp):
                etq <- etq + 1
        etq <- etq + 1
        etiqueta(Bloq)
        $.sig = etq



etiqueta(if_then_else(Exp, Bloq0, Bloq1)):
        $.prim = etq
        etiqueta(Exp)
        si es_desig(Exp):
                etq <- etq + 1
        etq <- etq + 1
```

```
        etiqueta(Bloq0)
        $.prim2 = etq
        etq <- etq + 1
        etiqueta(Bloq1)
        $.sig = etq

etiqueta(ins_call(Iden, Exp)):
        etiqueta(Exp)
        si es_desig(Exp):
                etq <- etq + 1
        etq <- etq + 1

etiqueta(while(Exp, Bloq)):
        $.prim = etq
        etiqueta(Exp)
        si es_desig(Exp):
                etq <- etq + 1
        etq <- etq + 1
        etiqueta(Bloq)
        etq <- etq + 1
        $.sig = etq

etiqueta(read(Exp)):
        etiqueta(Exp)
        t = ref!(Exp.tipo)
        si t == int || t == real || t == string
                etq <- etq + 2

etiqueta(write(Exp)):
        etiqueta(Exp)
        si es_desig(Exp):
                etq <- etq + 1
        etq <- etq + 1

etiqueta(ins_nl()):

etiqueta(new(Exp)):
        etiqueta(E)
        etq <- etq + 2


etiqueta(delete(Exp)):
        etiqueta(Exp)
        etq <- etq + 2

etiqueta(bloque(Exp)):
        etiqueta(Exp)
        etq <- etq
```

5.3. Etiquetado para las expresiones.

```
etiqueta(si_preal(LPRealOpt)):
        etiqueta(LPReal)

etiqueta(no_preal()): noop

etiqueta(muchos_preal(LPreal, Exp)):
        etiqueta(LPReal)
        etiqueta(Exp)

etiqueta(un_preal(Exp)):
        etiqueta(Exp)

etiqueta(exp_litEntero(N)):
        etq <- etq + 1

etiqueta(exp_litReal(N)):
        etq <- etq + 1

etiqueta(exp_litBoolTrue()):
        etq <- etq + 1

etiqueta(exp_litBoolFalse()):
        etq <- etq + 1

etiqueta(exp_litCadena(N)):
        etq <- etq + 1

etiqueta(exp_identificador(Id)):
        etq <- etq + 1

etiqueta(exp_null()):
        etq <- etq + 1
```

5.4. Operaciones

```
etiqueta(asig(Exp0,Exp1)):
        etiqueta(Exp0)
        etiqueta(Exp1)
        etq <- etq + 1
        if ref!(Exp1.tipo) == int & ref!(Exp0.tipo) == real
```

```
                if es_desig(Exp1)
                        etq <- etq + 1
                etq <- etq + 2
        else
                if es_desig(Exp1)
                        etq <- etq + 1

etiqueta(suma(Exp0,Exp1)):
        t0 = ref!(Exp0.tipo)
        t1 = ref!(Exp1.tipo)

        etiqueta(Exp0)
        if es_desig(Exp0)
                etq <- etq + 1
        if t0 == int & t1 == real
                etq <- etq + 1

        etiqueta(Exp1)
        if es_desig(Exp1)
                etq <- etq + 1
        else if t0==real & t1 ==int
                etq <- etq + 1

        etq <- etq + 1









etiqueta(resta(Exp0,Exp1)):
        t0 = ref!(Exp0.tipo)
        t1 = ref!(Exp1.tipo)

        etiqueta(Exp0)
        if es_desig(Exp0)
                etq <- etq + 1
```

```
        if t0 == int & t1 == real
                etq <- etq + 1

        etiqueta(Exp1)
        if es_desig(Exp1)
                etq <- etq + 1
        else if t0==real & t1 ==int
                etq <- etq + 1
        etq <- etq + 1

etiqueta(mul(Exp0,Exp1)):
        t0 = ref!(Exp0.tipo)
        t1 = ref!(Exp1.tipo)

        etiqueta(Exp0)
        if es_desig(Exp0)
                etq <- etq + 1
        if t0 == int & t1 == real
                etq <- etq + 1

        etiqueta(Exp1)
        if es_desig(Exp1)
                etq <- etq + 1
        else if t0==real & t1 ==int
                etq <- etq + 1
        etq <- etq + 1
```

```
etiqueta(div(Exp0,Exp1)):
        t0 = ref!(Exp0.tipo)
        t1 = ref!(Exp1.tipo)

        etiqueta(Exp0)
        if es_desig(Exp0)
```

```
                etq <- etq + 1
        if t0 == int & t1 == real
                etq <- etq + 1

        etiqueta(Exp1)
        if es_desig(Exp1)
                etq <- etq + 1
        else if t0==real & t1 ==int
                etq <- etq + 1
        etq <- etq + 1

etiqueta(mod(Exp0,Exp1)):
        t0 = ref!(Exp0.tipo)
        t1 = ref!(Exp1.tipo)

        etiqueta(Exp0)
        if es_desig(Exp0)
                etq <- etq + 1

        etiqueta(Exp1)
        if es_desig(Exp1)
                etq <- etq + 1

        etq <- etq + 1

etiqueta(and(Exp0,Exp1)):
        t0 = ref!(Exp0.tipo)
        t1 = ref!(Exp1.tipo)

        etiqueta(Exp0)
        if es_desig(Exp0)
                etq <- etq + 1

        etiqueta(Exp1)
        if es_desig(Exp1)
                etq <- etq + 1
        etq <- etq + 1



etiqueta(or(Exp0,Exp1)):
        t0 = ref!(Exp0.tipo)
        t1 = ref!(Exp1.tipo)

        etiqueta(Exp0)
        if es_desig(Exp0)
```

```
                etq <- etq + 1

        etiqueta(Exp1)
        if es_desig(Exp1)
                etq <- etq + 1
        etq <- etq + 1

etiqueta(mayor(Exp0,Exp1)):
        t0 = ref!(Exp0.tipo)
        t1 = ref!(Exp1.tipo)

        etiqueta(Exp0)
        if es_desig(Exp0)
                etq <- etq + 1
        if t0 == int & t1 == real
                etq <- etq + 1

        etiqueta(Exp1)
        if es_desig(Exp1)
                etq <- etq + 1
        else if t0==real & t1 ==int
                etq <- etq + 1
        etq <- etq + 1

etiqueta(menor(Exp0,Exp1)):
        t0 = ref!(Exp0.tipo)
        t1 = ref!(Exp1.tipo)

        etiqueta(Exp0)
        if es_desig(Exp0)
                etq <- etq + 1
        if t0 == int & t1 == real
                etq <- etq + 1

        etiqueta(Exp1)
        if es_desig(Exp1)
                etq <- etq + 1
        else if t0==real & t1 ==int
                etq <- etq + 1
        etq <- etq + 1


etiqueta(menorIgual(Exp0,Exp1)):
        t0 = ref!(Exp0.tipo)
        t1 = ref!(Exp1.tipo)
        etiqueta(Exp0)
        if es_desig(Exp0)
                etq <- etq + 1
```

```
        if t0 == int & t1 == real
                etq <- etq + 1
        etiqueta(Exp1)
        if es_desig(Exp1)
                etq <- etq + 1
        else if t0==real & t1 ==int
                etq <- etq + 1
        etq <- etq + 1

etiqueta(mayorIgual(Exp0,Exp1)):
        t0 = ref!(Exp0.tipo)
        t1 = ref!(Exp1.tipo)
        etiqueta(Exp0)
        if es_desig(Exp0)
                etq <- etq + 1
        if t0 == int & t1 == real
                etq <- etq + 1
        etiqueta(Exp1)
        if es_desig(Exp1)
                etq <- etq + 1
        else if t0==real & t1 ==int
                etq <- etq + 1
        etq <- etq + 1

etiqueta(desigual(Exp0,Exp1)):
        t0 = ref!(Exp0.tipo)
        t1 = ref!(Exp1.tipo)
        etiqueta(Exp0)
        if es_desig(Exp0)
                etq <- etq + 1
        if t0 == int & t1 == real
                etq <- etq + 1
        etiqueta(Exp1)
        if es_desig(Exp1)
                etq <- etq + 1
        else if t0==real & t1 ==int
                etq <- etq + 1
        etq <- etq + 1



etiqueta(igual(Exp0,Exp1)):
        t0 = ref!(Exp0.tipo)
        t1 = ref!(Exp1.tipo)
        etiqueta(Exp0)
        if es_desig(Exp0)
                etq <- etq + 1
```

```
        if t0 == int & t1 == real
                etq <- etq + 1
        etiqueta(Exp1)
        if es_desig(Exp1)
                etq <- etq + 1
        else if t0==real & t1 ==int
                etq <- etq + 1
        etq <- etq + 1

etiqueta(neg(Exp0)):
        t0 = ref!(Exp0.tipo)
        etiqueta(Exp0)
        if es_desig(Exp0)
                etq <- etq + 1
        etq <- etq + 1

etiqueta(not(Exp0)):
        t0 = ref!(Exp0.tipo)
        etiqueta(Exp0)
        if es_desig(Exp0)
                etq <- etq + 1
        etq <- etq + 1

etiqueta(acceso_array(Exp0,Exp1)):
        etiqueta(Exp0)
        etiqueta(Exp1)
        if es_desig(ref!(Exp0))
                etq <- etq + 1
        etq <- etq + 1

etiqueta(acceso_campo(Exp, Campo)):
        etiqueta(Exp)

etiqueta(acceso_puntero(Exp)):
        etiqueta(Exp)
```

6. Especificación del procesamiento de generación de código.

6.1. Generación de código para programa.

```
global procs = pila_vacia()

gen-cod(prog(Bloq)):
        gen-cod(Bloq)
```

```
gen-cod(Bloq(DecsOpt, InsOpt)):
        recolecta_procs(DecsOpt)
        gen-cod(InsOpt)
        emit stop()
        mientras(!es_vacia(procs))):
                sub = cima(sub_pendientes)
                desapila(sub_pendientes)
                let sub = dec_proc(Iden,Param,LDecs,Is) in
                emit desapilad(sub.nivel)
                recolecta_subs(LDecs)
                gen-cod(Ins)
                emit desactiva(sub.nivel,sub.tam)
                emid ir-ind()
                end let

gen-cod(si_decs()):
        recolecta_procs(LDecs)

gen-cod(no_dec()):
        skip

gen-cod(una_dec(Dec)):
        recolecta_procs(Dec)

gen-cod(muchas_decs(LDecs, Dec)):
        recolecta_procs(LDecs)
        recolecta_procs(Dec)

gen-cod(dec_var(T,Iden)):
        skip

gen-cod(dec_tipo(T,Iden)):
        skip
```

6.2. Generación de código para declaraciones.

```
gen-cod(dec_proc(Iden, PFormOpt, Bloq)):
        gen-cod(Bloq)
        emit desactiva(nivel, tam)
        emit ir_ind()
```

6.3. Generación de código para instrucciones.

```
gen-cod(si_ins(LIns)):
        gen-cod(LIns)

gen-cod(no_ins()):
        skip
```

```
gen-cod(una_ins(Ins)) =
        gen-cod(Ins)

gen-cod(muchas_ins(LIns, Ins)) =
        gen-cod(LIns)
        gen-cod(Ins)

gen-cod(ins_asig(Exp)):
        gen-cod(Exp)
        emit desecha()

gen-acc-val(Exp):
        si es_desig(Exp):
                v = emit fetch(r)
        sino:
                v = r

gen-cod(ins_if(Exp, Bloq)):
        gen-cod(Exp)
        gen-acc-val(Exp)
        emit ir-f($.sig)
        gen-cod(Bloq)

gen-cod(if_then_else(Exp, Bloq0, Bloq1)):
        gen-cod(Exp)
        gen-acc-val(Exp)
        emit ir-f($.sig)
        gen-cod(Bloq0)
        emit ir-a($.sig)
        gen-cod(Bloq1)

gen-cod(while(Exp, Bloq)):
        gen-cod(Exp)
        gen-acc-val(Exp)
        emit ir-f($.sig)
        gen-cod(Bloq)
        emit ir-a($.prim)



gen-cod(read(Exp)):
        t = ref!(Exp.tipo)
        r = gen-cod(Exp)
        emit leer_entrada_<t>
        store(r, t)

gen-cod(write(Exp)):
```

```
        r = gen-cod(Exp)
        gen-acc-val(Exp)
        emite mostrar_<v>

gen-cod(ins_nl()):
        skip

gen-cod(new(Exp)):
        emite store(gen-cod(Exp), emite alloc(ref!(Exp.tipo)))

dealloc(d, ), si d  -1. Error de ejecución si d = -1

gen-cod(delete(Exp)):
        d = gen-cod(Exp)
        si d != -1:
                emite dealloc(d, ref!(E.tipo))
        sino:
                ERROR

gen-cod(ins_call(string, PRealOpt)):
        let id = string.vinculo, id.vinculo = dec_proc(_, PFormOpt , _) in
                emit activa($.vinculo.nivel, $.vinculo.tam, $.dir_sig)
                gen-paso-pf(PFormOpt, PRealOpt)
                emit ir-a($.vinculo.dir_inic)
        end let


gen-cod(ins_bloque(Bloque)):
        gen-cod(Bloq)
```

6.4. Generación de codigo para expresiones

```
gen-cod(exp_litEntero(N)):
        emit apila_int(N)

gen-cod(exp_litReal(N)):
        emit apila_real(N)

gen-cod(exp_litBoolTrue()):
        emit apila_bool(True)

gen-cod(exp_litBoolFalse()):
        emit apila_bool(False)

gen-cod(exp_litCadena(N)):
        emit apila_string(N)

gen-cod(exp_identificador(Id)):
```

```
        if $.vinculo.nivel != 0:
                emit apilad($.vinculo.nivel)
                emit apila_int($.vinculo.dir)
                emit suma_int
                if $.vinculo == pform_ref(T, id)
                        emit apila_ind

        sino:
                emit apila_int($.vinculo.dir)

gen-cod(exp_null()):
        emit apila_int(-1)))

gen-acc-val(Exp):
        if es_desig(ref!(Exp))
                emit apila_ind()

gen-cod(asig(Exp0,Exp1)):
        gen-cod(Exp0)
        gen-cod(Exp1)
        if ref!(Exp1.tipo) == int & ref!(Exp0.tipo) == real
                if es_desig(Exp1)
                        emit apila_ind
                emit int2real
                emit desapila_ind
        else
                if es_desig(Exp1)
                        emit mueve(ref!(Exp0.tipo).tam)
                else
                        emit desapila_ind

gen-cod(suma(Exp0,Exp1)):
        t0 = ref!(Exp0.tipo)
        t1 = ref!(Exp1.tipo)
        if t0 == int & t1 == real
                gen-cod(Exp0)
                gen-acc-val(Exp0)
                emit int2real
```

```
                gen-cod(Exp1)
                gen-acc-val(Exp1)
                emit suma_real
        else
                gen-cod(Exp0)
                gen-acc-val(Exp0)  //si es designador apila_ind sino nada
                gen-cod(Exp1)
                gen-acc-val(Exp1)

        if t0 == int & t1 == int
                emit suma_int
        else if t0 == real & t1 = real
                emit suma_real
        else if t0==real & t1 ==int
                emit int2real
                emit suma_real

gen-cod(resta(Exp0,Exp1)):
        t0 = ref!(Exp0.tipo)
        t1 = ref!(Exp1.tipo)
        if t0 == int & t1 == real
                gen-cod(Exp0)
                gen-acc-val(Exp0)
                emit int2real
                gen-cod(Exp1)
                gen-acc-val(Exp1)
                emit resta_real
        else
                gen-cod(Exp0)
                gen-acc-val(Exp0)  //si es designador apila_ind sino nada
                gen-cod(Exp1)
                gen-acc-val(Exp1)
        if t0 == int & t1 == int
                emit resta_int
        else if t0 == real & t1 = real
                emit resta_real
        else if t0==real & t1 ==int
                emit int2real
                emit resta_real


gen-cod(mul(Exp0,Exp1)):
        t0 = ref!(Exp0.tipo)
        t1 = ref!(Exp1.tipo)
        if t0 == int & t1 == real
                gen-cod(Exp0)
                gen-acc-val(Exp0)
                emit int2real
```

```
                gen-cod(Exp1)
                gen-acc-val(Exp1)
                emit mul_real
        else
                gen-cod(Exp0)
                gen-acc-val(Exp0)  //si es designador apila_ind sino nada
                gen-cod(Exp1)
                gen-acc-val(Exp1)
        if t0 == int & t1 == int
                emit mul_int
        else if t0 == real & t1 = real
                emit mul_real
        else if t0==real & t1 ==int
                emit int2real
                emit mul_real

gen-cod(div(Exp0,Exp1)):
        t0 = ref!(Exp0.tipo)
        t1 = ref!(Exp1.tipo)
        if t0 == int & t1 == real
                gen-cod(Exp0)
                gen-acc-val(Exp0)
                emit int2real
                gen-cod(Exp1)
                gen-acc-val(Exp1)
                emit div_real
        else
                gen-cod(Exp0)
                gen-acc-val(Exp0)  //si es designador apila_ind sino nada
                gen-cod(Exp1)
                gen-acc-val(Exp1)
        if t0 == int & t1 == int
                emit div_int
        else if t0 == real & t1 = real
                emit div_real
        else if t0==real & t1 ==int
                emit int2real
                emit div_real



gen-cod(mod(Exp0,Exp1)):
        gen-cod(Exp0)
        gen-acc-val(Exp0)
        gen-cod(Exp1)
        gen-acc-val(Exp1)
        emit mod
```

```
gen-cod(and(Exp0,Exp1)):
        gen-cod(Exp0)
        gen-acc-val(Exp0)
        gen-cod(Exp1)
        gen-acc-val(Exp1)
        emit and

gen-cod(or(Exp0,Exp1)):
        gen-cod(Exp0)
        gen-acc-val(Exp0)
        gen-cod(Exp1)
        gen-acc-val(Exp1)
        emit or

gen-cod(mayor(Exp0,Exp1)):
        t0 = ref!(Exp0.tipo)
        t1 = ref!(Exp1.tipo)
        if t0 == int & t1 == real
                gen-cod(Exp0)
                gen-acc-val(Exp0)
                emit int2real
                gen-cod(Exp1)
                gen-acc-val(Exp1)
                emit mayor_real
        else
                gen-cod(Exp0)
                gen-acc-val(Exp0)  //si es designador apila_ind sino nada
                gen-cod(Exp1)
                gen-acc-val(Exp1)
        if t0 == t1
                if t1 ==real
                        emit mayor_real
                else if t1 == int
                        emit mayor_int
                else if t1 == bool
                        emit mayor_bool
                else if t1 == string
                        emit mayor_string
        else if t0==real & t1 ==int
                emit int2real
                emit mayor_real
gen-cod(menor(Exp0,Exp1)):
        t0 = ref!(Exp0.tipo)
        t1 = ref!(Exp1.tipo)
        if t0 == int & t1 == real
                gen-cod(Exp0)
                gen-acc-val(Exp0)
                emit int2real
```

```
                gen-cod(Exp1)
                gen-acc-val(Exp1)
                emit menor_real
        else
                gen-cod(Exp0)
                gen-acc-val(Exp0)  //si es designador apila_ind sino nada
                gen-cod(Exp1)
                gen-acc-val(Exp1)
        if t0 == t1
                if t1 ==real
                        emit menor_real
                else if t1 == int
                        emit menor_int
                else if t1 == bool
                        emit menor_bool
                else if t1 == string
                        emit menor_string
        else if t0==real & t1 ==int
                emit int2real
                emit menor_real

gen-cod(menorIgual(Exp0,Exp1)):
        t0 = ref!(Exp0.tipo)
        t1 = ref!(Exp1.tipo)
        if t0 == int & t1 == real
                gen-cod(Exp0)
                gen-acc-val(Exp0)
                emit int2real
                gen-cod(Exp1)
                gen-acc-val(Exp1)
                emit menorIgual_real
        else
                gen-cod(Exp0)
                gen-acc-val(Exp0)  //si es designador apila_ind sino nada
                gen-cod(Exp1)
                gen-acc-val(Exp1)
        if t0 == t1
                if t1 ==real
                        emit menorIgual_real
                else if t1 == int
                        emit menorIgual_int
                else if t1 == bool
                        emit menorIgual_bool
                else if t1 == string
                        emit menorIgual_string
        else if t0==real & t1 ==int
                emit int2real
                emit menorIgual_real
```

```
gen-cod(mayorIgual(Exp0,Exp1)):
        t0 = ref!(Exp0.tipo)
        t1 = ref!(Exp1.tipo)
        if t0 == int & t1 == real
                gen-cod(Exp0)
                gen-acc-val(Exp0)
                emit int2real
                gen-cod(Exp1)
                gen-acc-val(Exp1)
                emit mayorIgual_real
        else
                gen-cod(Exp0)
                gen-acc-val(Exp0)  //si es designador apila_ind sino nada
                gen-cod(Exp1)
                gen-acc-val(Exp1)
        if t0 == t1
                if t1 ==real
                        emit mayorIgual_real
                else if t1 == int
                        emit mayorIgual_int
                else if t1 == bool
                        emit mayorIgual_bool
                else if t1 == string
                        emit mayorIgual_string
        else if t0==real & t1 ==int
                emit int2real
                emit mayorIgual_real
```

```
gen-cod(desigual(Exp0,Exp1)):
        t0 = ref!(Exp0.tipo)
        t1 = ref!(Exp1.tipo)
        if t0 == int & t1 == real
                gen-cod(Exp0)
                gen-acc-val(Exp0)
                emit int2real
```

```
                gen-cod(Exp1)
                gen-acc-val(Exp1)
                emit desigual_real
        else
                gen-cod(Exp0)
                gen-acc-val(Exp0)  gen-cod(Exp1)
                gen-acc-val(Exp1)
        if t0 == t1
                if t1 ==real
                        emit desigual_real
                else if t1 == int
                        emit desigual_int
                else if t1 == bool
                        emit desigual_bool
                else if t1 == string
                        emit desigual_string
                else if t1 == puntero
                        emit desigual_puntero
                else if t1 == null
                        emit desigual_null
        else if t0==real & t1 ==int
                emit int2real
                emit desigual_real
        else if t0 == puntero & t1 == null || t1 == null & t1 == puntero
                emit desigual_pn_null
```

```
gen-cod(igual(Exp0,Exp1)):
        t0 = ref!(Exp0.tipo)
        t1 = ref!(Exp1.tipo)
        if t0 == int & t1 == real
                gen-cod(Exp0)
                gen-acc-val(Exp0)
                emit int2real
```

```
                gen-cod(Exp1)
                gen-acc-val(Exp1)
                emit igual_real
        else
                gen-cod(Exp0)
                gen-acc-val(Exp0)  //si es designador apila_ind sino nada
                gen-cod(Exp1)
                gen-acc-val(Exp1)
        if t0 == t1
                if t1 ==real
                        emit igual_real
                else if t1 == int
                        emit igual_int
                else if t1 == bool
                        emit igual_bool
                else if t1 == string
                        emit igual_string
                else if t1 == puntero
                        emit igual_puntero
                else if t1 == null
                        emit igual_null
        else if t0==real & t1 ==int
                emit int2real
                emit igual_real
        else if t0 == puntero & t1 == null || t1 == null & t1 == puntero
                emit igual_pn_null

gen-cod(neg(Exp0)):
        t0 = ref!(Exp0.tipo)
        gen-cod(Exp0)
        gen-acc-val(Exp0)
        if t0 == int
                emit neg_int
        else if t0 == real
                emit neg_real

gen-cod(not(Exp0)):
        gen-cod(Exp0)
        gen-acc-val(Exp0)
        emit not

gen-cod(acceso_array(Exp0,Exp1)):
        gen-cod(Exp0)
        gen-cod(Exp1)
        gen-acc-val(Exp1)
        sea ref!(Exp1.tipo) = array (Tipo, d)
                emit apila_int(Tipo.tam)
        emit mul_int
```

```
        emit suma_int

gen-cod(acceso_campo(Exp, Campo)):
        gen-cod(Exp)
        let ref!(Exp.tipo) = struct(LCampos)
                emit apila_int(desplazamiento(LCampos, Campo)
        emit suma_int

gen-cod(acceso_puntero(Exp)):
        gen-cod(Exp)
        emit apila_ind()
```