Práctica. Cuarta fase
Finalización del procesador para Tiny

Grupo 18

Alumnos:

Jorge Torres Ruiz
Alvaro Montes Anacona
Daniel López Escobar
Ignacio García Fernández

1. Especificación del procesamiento de vinculación

```
var ts // La tabla de símbolos
var noop = null

contiene_inserta(id):
    if contiene(ts,Id) then
        error
    else
        inserta(ts,Id,$)
    end if

vinculaId(id):
    $.vinculo = vinculoDe(ts,Id)
    if $.vinculo == ⊥ then
        error
    end if

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

vincula(bloque(decs, intrs)):
    abreAmbito(ts)
    recolectaDecs(decs)
    vincula(intrs)
```

```
    cierraAmbito(ts)

recolectaDecs(si_decs(decs)):
    recolectaDecs1(decs)
    recolectaDecs2(decs)

recolectaDecs(no_decs()):
    noop

recolectaDecs1(mas_decs(decs, dec)):
    recolectaDecs1(decs)
    recolectaDec1(dec)

recolectaDecs2(mas_decs(decs, dec)):
    recolectaDecs2(decs)
    recolectaDec2(dec)

recolectaDecs1(una_dec(dec)):
    recolectaDec1(dec)

recolectaDecs2(una_dec(dec)):
    recolectaDec2(dec)

recolectaDec1(dec_var(tipo, id)):
    vincula1(tipo)
    contiene_inserta(id)

recolectaDec2(dec_var(tipo, id)):
    vincula2(tipo)

recolectaDec1(dec_tipo(tipo, id)):
    vincula1(tipo)
    contiene_inserta(id)

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

recolectaDec1(dec_proc(id, pforms, bloque)):
    contiene_inserta(id)
    abreAmbito(ts)
    recolectaPForms(pforms)
    vincula(bloque)
    cierraAmbito(ts)
```

```
recolectaDec2(dec_proc(id, pforms, bloque)):
    noop

recolectaPForms(si_pforms(pfroms)):
    recolectaPForms(pforms)

recolectaPForms(no_pforms()):
    noop

recolectaPForms(mas_pforms(pforms, pform)):
    recolectaPForms(pfroms)
    recolectaPForm(pform)

recolectaPForms(una_pform(pform)):
    recolectaPForm(pform)

recolectaPForm(pform(tipo, ref, id)):
    vincula(tipo)
    contiene_inserta(id)

// si_ref() No afecta

// no_ref() No afecta

vincula1(t_iden(id)):
    vinculaId(id)

vincula2(t_iden(id)):
    noop

vincula1(t_string()):
    noop

vincula2(t_string()):
    noop

vincula1(t_int()):
    noop

vincula2(t_int()):
    noop
vincula1(t_bool()):
```

```
    noop

vincula2(t_bool()):
    noop

vincula1(t_real()):
    noop

vincula2(t_real()):
    noop

vincula1(t_array(tipo, ent)):
    vincula1(tipo)

vincula2(t_array(tipo, ent)):
    vincula2(tipo)

vincula1(t_punter(tipo)):
    if tipo != t_iden(_) then
        vincula1(tipo)
    end if

vincula2(t_punter(tipo)):
    if tipo == t_iden(_) then
        vincula1(tipo)
    else
        vincula2(tipo)
    end if

vincula(t_struct(camposs)):
    vincula(camposs)

vincula2(t_struct(camposs)):
    vincula2(camposs)

vincula(mas_cmp_s(camposs, campos)):
    vincula(camposs)
    vincula(campos)

vincula2(mas_cmp_s(camposs, campos)):
    vincula2(camposs)
    vincula2(campos)
vincula(un_cmp_s(campos)):
```

```
    vincula(campos)

vincula2(un_cmp_s(campos)):
    vincula2(campos)

vincula(cmp_s(tipo, id)):
    vincula(tipo)

vincula2(cmp_s(tipo, id)):
    vincula2(tipo)

vincula(si_intrs(intrs)):
    vincula(intrs)

vincula(no_intrs()):
    noop

vincula(mas_intrs(intrs, intr)):
    vincula(intrs)
    vincula(intr)

vincula(una_intr(intr)):
    vincula(intr)

vincula(i_eval(exp)):
    vincula(exp)

vincula(i_if(exp, bloque, ielse)):
    vincula(exp)
    vincula(bloque)
    vincula(ielse)

vincula(i_while(exp, bloque)):
    vincula(exp)
    vincula(bloque)

vincula(i_read(exp)):
    vincula(exp)

vincula(i_write(exp)):
    vincula(exp)

vincula(i_nl()):
```

```
    noop

vincula(i_new(exp)):
    vincula(exp)

vincula(i_delete(exp)):
    vincula(exp)

vincula(i_call(id, preals)):
    vinculaId(id)
    vincula(preals)

vincula(i_prog(bloque)):
    vincula(bloque)

vincula(si_else(bloque)):
    vincula(bloque)

vincula(no_else()):
    noop

vincula(si_preals(preals)):
    vincula(preals)

vincula(no_preals()):
    noop

vincula(mas_preals(preals, exp)):
    vincula(preals)
    vincula(exp)

vincula(un_preals(exp)):
    vincula(exp)

vincula(e_asig(opnd0, opnd1)):
    vincula(opnd0)
    vincula(opnd1)

vincula(e_comp(opnd0, opnd1)):
    vincula(opnd0)
    vincula(opnd1)

vincula(e_dist(opnd0, opnd1)):
```

```
    vincula(opnd0)
    vincula(opnd1)

vincula(e_lt(opnd0, opnd1)):
    vincula(opnd0)
    vincula(opnd1)

vincula(e_gt(opnd0, opnd1)):
    vincula(opnd0)
    vincula(opnd1)

vincula(e_leq(opnd0, opnd1)):
    vincula(opnd0)
    vincula(opnd1)

vincula(e_geq(opnd0, opnd1)):
    vincula(opnd0)
    vincula(opnd1)

vincula(e_suma(opnd0, opnd1)):
    vincula(opnd0)
    vincula(opnd1)

vincula(e_resta(opnd0, opnd1)):
    vincula(opnd0)
    vincula(opnd1)

vincula(e_and(opnd0, opnd1)):
    vincula(opnd0)
    vincula(opnd1)

vincula(e_or(opnd0, opnd1)):
    vincula(opnd0)
    vincula(opnd1)

vincula(e_mul(opnd0, opnd1)):
    vincula(opnd0)
    vincula(opnd1)

vincula(e_div(opnd0, opnd1)):
    vincula(opnd0)
    vincula(opnd1)
```

```
vincula(e_porcentaje(opnd0, opnd1)):
    vincula(opnd0)
    vincula(opnd1)

vincula(e_negativo(opnd)):
    vincula(opnd)

vincula(e_negado(opnd)):
    vincula(opnd)

vincula(e_indexado(opnd0, opnd1)):
    vincula(opnd0)
    vincula(opnd1)

vincula(e_campo(opnd, id)):
    vincula(opnd)
    // Aqui no habria que hacer vincula, no? porque lo haria
comprobando tipos?

vincula(e_puntero(opnd)):
    vincula(opnd)

vincula(e_lit_ent(num)):
    noop

vincula(e_lit_real(num)):
    noop

vincula(e_true()):
    noop

vincula(e_false()):
    noop

vincula(e_string(string)):
    noop

vincula(e_iden(id)):
    vinculaId(id)

vincula(e_null()):
    noop
```

2. Especificación del procesamiento de pre-tipado

```
var conjunto

pretipado(prog(bloque)):
    pretipado(bloque)

pretipado(bloque(decs, intrs)):
    pretipado(decs)
    pretipado(intrs)

pretipado(si_decs(decs)):
    pretipado(decs)

pretipado(no_decs()):
    noop

pretipado(mas_decs(decs, dec)):
    pretipado(decs)
    pretipado(dec)

pretipado(una_dec(dec)):
    pretipado(dec)

pretipado(dec_var(tipo, id)):
    pretipado(tipo)

pretipado(dec_tipo(tipo, id)):
    pretipado(tipo)

pretipado(dec_proc(id, pforms, bloque)):
    pretipado(pforms)
    pretipado(bloque)

pretipado(si_pforms(pfroms)):
    pretipado(pforms)
```

```
pretipado(no_pforms()):
    noop

pretipado(mas_pforms(pforms, pform)):
    pretipado(pforms)
    pretipado(pform)

pretipado(una_pform(pform)):
    pretipado(pform)

pretipado(pform(tipo, ref, id)):
    pretipado(tipo)

// si_ref() no aplica

// no_ref() no aplica

pretipado(t_iden(id)):
    if $.vinculo != dec_tipo(_, _) then
        error
    end if

pretipado(t_string()):
    noop

pretipado(t_int()):
    noop

pretipado(t_bool()):
    noop

pretipado(t_real()):
    noop

pretipado(t_array(tipo, ent)):
    pretipado(tipo)
```

```
    if int(ent) <= 0 then // El tamaño de un array debe ser mayor que 0
        error
    end if

pretipado(t_punter(tipo)):
    pretipado(tipo)

pretipado(t_struct(camposs)):
    abreAmbito(conjunto) // De esta forma cada struct tiene sus propios
campos
    pretipado(camposs)
    cierraAmbito(conjunto)

pretipado(mas_cmp_s(camposs, campos)):
    pretipado(camposs)
    pretipado(campos)

pretipado(un_cmp_s(campos)):
    pretipado(campos)

pretipado(cmp_s(tipo, id)):
    pretipado(tipo)
    if conjunto.contiene(id) then // El campo ya existe
        error
    else
        conjunto.inserta(id) // Inserta el campo en el conjunto
    end if

pretipado(si_intrs(intrs)):
    pretipado(intrs)

pretipado(no_intrs()):
    noop

pretipado(mas_intrs(intrs, intr)):
    pretipado(intrs)
```

```
    pretipado(intr)

pretipado(una_intr(intr)):
    pretipado(intr)

pretipado(i_eval(exp)):
    noop

pretipado(i_if(exp, bloque, ielse)):
    pretipado(bloque)
    pretipado(ielse)

pretipado(i_while(exp, bloque)):
    pretipado(bloque)

pretipado(i_read(exp)):
    noop

pretipado(i_write(exp)):
    noop

pretipado(i_nl()):
    noop

pretipado(i_new(exp)):
    noop

pretipado(i_delete(exp)):
    noop

pretipado(i_call(id, preals)):
    noop

pretipado(i_bloque(bloque)):
    pretipado(bloque)
```

```
pretipado(si_else(bloque)):
    pretipado(bloque)

pretipado(no_else()):
    noop

// si_preals(preals) no aplica
// no_preals() no aplica
// mas_preals(preals, exp) no aplica
// un_preals(exp) no aplica
// e_asig(opnd0, opnd1) no aplica
// e_comp(opnd0, opnd1) no aplica
// e_dist(opnd0, opnd1) no aplica
// e_lt(opnd0, opnd1) no aplica
// e_gt(opnd0, opnd1) no aplica
// e_leq(opnd0, opnd1) no aplica
// e_geq(opnd0, opnd1) no aplica
// e_suma(opnd0, opnd1) no aplica
// e_resta(opnd0, opnd1) no aplica
// e_and(opnd0, opnd1) no aplica
// e_or(opnd0, opnd1) no aplica
// e_mul(opnd0, opnd1) no aplica
// e_div(opnd0, opnd1) no aplica
// e_porcentaje(opnd0, opnd1) no aplica
// e_negativo(opnd) no aplica
// e_negado(opnd) no aplica
// e_indexado(opnd0, opnd1) no aplica
// e_campo(opnd, id) no aplica
// e_puntero(opnd) no aplica
// e_lit_ent(num) no aplica
// e_lit_real(num) no aplica
// e_true() no aplica
// e_false() no aplica
// e_string(string) no aplica
// e_iden(id) no aplica
// e_null() no aplica
```

3. Especificación del procesamiento de comprobación de tipos.

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

tipado(bloque(decs, intrs)):
    tipado(decs)
    tipado(intrs)
    $.tipo = ambos_ok(decs.tipo, intrs.tipo)

tipado(si_decs(decs)):
    tipado(decs)
    $.tipo = decs.tipo

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

tipado(mas_decs(decs, dec)):
    tipado(decs)
    tipado(dec)
    if (dec.tipo == ERROR || decs.tipo == ERROR) then
        $.tipo = ERROR
    else
        $.tipo = OK
    end if

tipado(una_dec(dec)):
    tipado(dec)
    if (dec.tipo == ERROR) then
        $.tipo = ERROR
    else
        $.tipo = OK
    end if


tipado(dec_var(tipo, id)):
```

```
    $.tipo = tipo

tipado(dec_tipo(tipo, id)):
    $.tipo = tipo

tipado(dec_proc(id, pforms, bloque)):
    tipado(bloque)
    $.tipo = bloque.tipo

// si_pforms(pfroms)
// no_pforms()
// mas_pforms(pforms, pform)
// una_pform(pform)
// pform(tipo, ref, id)
// si_ref()
// no_ref()
// t_iden(id)
// t_string()
// t_int()
// t_bool()
// t_real()
// t_array(tipo, ent)
// t_punter(tipo)
// t_struct(camposs)
// mas_cmp_s(camposs, campos)
// un_cmp_s(campos)
// cmp_s(tipo, id)

tipado(si_intrs(intrs)):
    tipado(intrs)
    $.tipo = intrs.tipo

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

tipado(mas_intrs(intrs, intr)):
```

```
    tipado(intrs)
    tipado(intr)
    $.tipo = ambos_ok(intrs.tipo, intr.tipo)

tipado(una_intr(intr)):
    tipado(intr)
    $.tipo = intr.tipo

tipado(i_eval(exp)):
    tipado(exp)
    if exp.tipo == ERROR then
        $.tipo = ERROR
    else
        $.tipo = OK
    end if

tipado(i_if(exp, bloque, ielse)):
    tipado(exp)
    tipoExp = ref!(exp.tipo)
    if tipoExp != t_bool() then
        error()
    end if
    tipado(bloque)
    tipado(ielse)
    if tipoExp == t_bool() && bloque.tipo == OK && ielse.tipo == OK
then
        $.tipo = OK
    else
        $.tipo = ERROR
    end if

tipado(i_while(exp, bloque)):
    tipado(exp)
    tipoExp = ref!(exp.tipo)
    if tipoExp != t_bool() then
        error()
```

```
    end if
    tipado(bloque)
    tipoExp = ref!(exp.tipo)
    if tipoExp == t_bool() && bloque.tipo == OK then
        $.tipo = OK
    else
        $.tipo = ERROR
    end if

tipado(i_read(exp)):
    tipado(exp)
    tipoExp = ref!(exp.tipo)
    if (tipoExp == t_int() || tipoExp == t_real() || tipoExp ==
t_string()) && asignable(exp) then
        $.tipo = OK
    else
        $.tipo = ERROR
    end if

tipado(i_write(exp)):
    tipado(exp)
    tipoExp = ref!(exp.tipo)
    if tipoExp == t_int() || tipoExp == t_real() || tipoExp == t_bool()
|| tipoExp == t_string() then
        $.tipo = OK
    else
        $.tipo = ERROR
    end if

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

tipado(i_new(exp)):
    tipado(exp)
    if ref!(exp.tipo) == t_punter() then
        $.tipo = OK
    else
```

```
        $.tipo = ERROR
    end if

tipado(i_delete(exp)):
    tipado(exp)
    if ref!(exp.tipo) == t_punter() then
        $.tipo = OK
    else
        $.tipo = ERROR
    end if

tipado(i_call(id, preals)):
    v = $.vinculo
    if v == dec_proc(_, pforms, _) then
        $.tipo = son_parametros_compatibles(pforms, preals)
    else
        $.tipo = ERROR
    end if

son_parametros_compatibles(no_pforms(), no_preals()):
    return OK

son_parametros_compatibles(no_pforms(), si_preals(_)):
    return ERROR

son_parametros_compatibles(si_pforms(_), no_preals()):
    return ERROR

son_parametros_compatibles(si_pforms(pforms), si_preals(preals)):
    if (len(pforms) != len(preals)) then
        return ERROR
    else
        return son_parametros_compatibles(pforms, preals)
    end if

len(mas_pforms(pforms, pform)):
```

```
    return len(pforms) + 1

len(una_pform(pform)):
    return 1

len(mas_preals(preals, exp)):
    return len(preals) + 1

len(un_preal(exp)):
    return 1

son_parametros_compatibles(mas_pforms(pforms, pform),
mas_preals(preals, exp)):
    t0 = son_parametros_compatibles(pforms, preals)
    t1 = es_parametro_compatible(pform, exp)
    return ambos_ok(t0, t1)

son_parametros_compatibles(una_pform(pform), un_preal(exp)):
    return es_parametro_compatible(pform, exp)

es_parametro_compatible(pform(tipo, ref, id), exp):
    tipado(exp)
    t0, t1 = ref!(exp.tipo), ref!(tipo)
    if ref == si_ref() && asignable(exp) then
        error("Para pasar por referencia, el argumento tiene que ser
una variable")
        return ERROR
    else if tipo_asig(t0, t1) == OK then // falta hacer tipo_asig, está
arriba, pero solo comprueba si son iguales (en este caso hay que seguir
las normas de las asignaciones)
        if ref == si_ref() && t1 == t_real() && t0 != t_real() then
            error("El tipo de la variable a la que se le pasa por
referencia tiene que coincidir con el tipo del argumento")
            return ERROR
        else
            return OK
        end if
```

```
    else
        error("El tipo del argumento tiene que ser compatible con el
tipo del parametro formal")
        return ERROR
    end if

tipado(i_bloque(bloque)):
    tipado(bloque)
    $.tipo = bloque.tipo

tipado(si_else(bloque)):
    tipado(bloque)
    $.tipo = bloque.tipo

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

// si_preals(preals)
// no_preals()
// mas_preals(preals, exp)
// un_preals(exp)

tipado(e_asig(opnd0, opnd1)):
    tipado(opnd0)
    tipado(opnd1)
    t0, t1 = ref!(opnd0.tipo), ref!(opnd1.tipo)
    if asignable(opnd0) && compatibles(t0, t1) then
        $.tipo = t0
    else
        aviso_error(t0, t1)
        $.tipo = ERROR
    end if

tipado_comp(opnd0, opnd1):
    tipado(opnd0)
    tipado(opnd1)
```

```
    t0, t1 = ref!(opnd0.tipo), ref!(opnd1.tipo)
    if ((t0 == t_punter() || t0 == NULL) && (t1 == t_punter() || t1 ==
NULL)) ||
        (t0 == t_string() && t1 == t_string()) ||
        (t0 == t_bool() && t1 == t_bool()) ||
        ((t0 == t_int() || t0 == t_real()) && (t1 == t_int() || t1 ==
t_real())) then
        $.tipo = t_bool()
    else
        aviso_error(t0, t1)
        $.tipo = ERROR
    end if

tipado_comp_ord(opnd0, opnd1):
    tipado(opnd0)
    tipado(opnd1)
    t0, t1 = ref!(opnd0.tipo), ref!(opnd1.tipo)
    if (t0 == t_string() && t1 == t_string()) ||
        (t0 == t_bool() && t1 == t_bool()) ||
        ((t0 == t_int() || t0 == t_real()) && (t1 == t_int() || t1 ==
t_real())) then
        $.tipo = t_bool()
    else
        aviso_error(t0, t1)
        $.tipo = ERROR
    end if

tipado(e_comp(opnd0, opnd1)):
    tipado_comp(opnd0, opnd1)

tipado(e_dist(opnd0, opnd1)):
    tipado_comp(opnd0, opnd1)

tipado(e_lt(opnd0, opnd1)):
    tipado_comp_ord(opnd0, opnd1)

tipado(e_gt(opnd0, opnd1)):
```

```
    tipado_comp_ord(opnd0, opnd1)

tipado(e_leq(opnd0, opnd1)):
    tipado_comp_ord(opnd0, opnd1)

tipado(e_geq(opnd0, opnd1)):
    tipado_comp_ord(opnd0, opnd1)

    tipado_arit(opnd0, opnd1):
    tipado(opnd0)
    tipado(opnd1)
    t0, t1 = ref!(opnd0.tipo), ref!(opnd1.tipo)
    if (t0 == t_int() || t0 == t_real()) && (t1 == t_int() || t1 ==
t_real()) then
        $.tipo = (t0 == t_real() || t1 == t_real()) ? t_real() :
t_int()
    else
        aviso_error(t0, t1)
        $.tipo = ERROR
    end if

tipado(e_suma(opnd0, opnd1)):
    tipado_arit(opnd0, opnd1)

tipado(e_resta(opnd0, opnd1)):
    tipado_arit(opnd0, opnd1)

tipado_and_or(opnd0, opnd1):
    tipado(opnd0)
    tipado(opnd1)
    t0, t1 = ref!(opnd0.tipo), ref!(opnd1.tipo)
    if t0 == t_bool() && t1 == t_bool() then
        $.tipo = t_bool()
    else
        aviso_error(t0, t1)
        $.tipo = ERROR
    end if
```

```
tipado(e_and(opnd0, opnd1)):
    tipado_and_or(opnd0, opnd1)

tipado(e_or(opnd0, opnd1)):
    tipado_and_or(opnd0, opnd1)

tipado(e_mul(opnd0, opnd1)):
    tipado_arit(opnd0, opnd1)

tipado(e_div(opnd0, opnd1)):
    tipado_arit(opnd0, opnd1)

tipado(e_porcentaje(opnd0, opnd1)):
    tipado(opnd0)
    tipado(opnd1)
    t0, t1 = ref!(opnd0.tipo), ref!(opnd1.tipo)
    if t0 == t_int() && t1 == t_int() then
        $.tipo = t_int()
    else
        aviso_error(t0, t1)
        $.tipo = ERROR
    end if

tipado(e_negativo(opnd)):
    tipado(opnd)
    tipo = ref!(opnd.tipo)
    if tipo == t_int() || tipo == t_real() then
        $.tipo = tipo
    else
        $.tipo = ERROR
    end if

tipado(e_negado(opnd)):
    tipado(opnd)
    tipo = ref!(opnd.tipo)
```

```
    if tipo == t_bool() then
        $.tipo = t_bool()
    else
        $.tipo = ERROR
    end if

tipado(e_indexado(opnd0, opnd1)):
    tipado(opnd0)
    tipado(opnd1)
    t0, t1 = ref!(opnd0.tipo), ref!(opnd1.tipo)
    if t0 == t_array(tb, _) && t1 == t_int() then
        $.tipo = tb
    else
        aviso_error(t0, t1)
        $.tipo = ERROR
    end if

tipado(e_campo(opnd, id)):
    tipado(opnd)
    tipo = ref!(opnd.tipo)

    if tipo == t_struct(campos) then
        $.tipo = busquedaCampo(id, campos)
    else
        error("Se esta intentando acceder a un campo de un tipo no
estructurado")
        $.tipo = ERROR
    end if

busquedaCampo(idCampo, mas_cmp_s(camposs, cmp_s(tipo, id))):
    if idCampo == id then
        return tipo
    else
        return busquedaCampo(id, camposs)
    end if
```

```
busquedaCampo(idCampo, un_cmp_s(cmp_s(tipo, id))):
    if idCampo == id then
        return tipo
    else
        error("Campo no definido en el struct")
        return ERROR
    end if

tipado(e_puntero(opnd)):
    tipado(opnd)
    tipo = ref!(opnd.tipo)
    if tipo == t_punter(tb) then
        $.tipo = tb
    else
        error("Se esta intentando desreferenciar un puntero a un tipo
no puntero")
        $.tipo = ERROR
    end if

tipado(e_lit_ent(num)):
    $.tipo = t_int()

tipado(e_lit_real(num)):
    $.tipo = t_real()

tipado(e_true()):
    $.tipo = t_bool()

tipado(e_false()):
    $.tipo = t_bool()

tipado(e_string(string)):
    $.tipo = t_string()

tipado(e_iden(id)):
    if $.vinculo != dec_var(_, _) then
```

```
        error("No es una variable")
        $.tipo = ERROR
    else
        $.tipo = $.vinculo.tipo
    end if

tipado(e_null()):
    $.tipo = NULL

compatibles(t1, t2):
    C = {t1 = t2}
    return unificables(t1, t2)

unificables(t1, t2):
    t1p, t2p = ref!(t1), ref!(t2)

    if  (t1p == INT && t2p == INT) ||
        (t1p == REAL && (t2p == INT || t2p == REAL)) ||
        (t1p == BOOL && t2p == BOOL) ||
        (t1p == STRING && t2p == STRING) then
        return true
    else if t1p == t_array(t1a, n1) && t2p == t_array(t2a, n2) then
        return n1 == n2 && son_unificables(t1a, t2a)
    else if t1p == t_struct(camposs1) && t2p == t_struct(camposs2) then
        return len(camposs1) == len(camposs2) &&
son_campos_unificables(camposs1, camposs2)
    else if t1p == t_punter(t1a) && t2p == NULL then
        return true
    else if t1p == t_punter(t1a) && t2p == t_punter(t2a)
        return son_unificables(t1a, t2a)
    else
        return false
    end if

son_unificables(t1, t2):
    if C.contiene(t1 = t2) then
```

```
        return true
    else
        C.add(t1 = t2)
        return unificables(t1, t2)
    end if

son_campos_unificables(mas_cmp_s(camposs1, campos1),
mas_cmp_s(camposs2, campos2)):
    return son_campos_unificables(camposs1, camposs2) &&
son_unificables(campos1, campos2)

son_campos_unificables(un_cmp_s(campos1), un_cmp_s(campos2)):
    return son_campos_unificables(campos1, campos2)

son_campos_unificables(cmp_s(t1, _), cmp_s(t2, _)):
    t1p, t2p = ref!(t1), ref!(t2)
    return son_unificables(t1, t2)

asignable(exp):
    return  exp == e_iden(_) || exp == e_campo(_, _) ||
            exp == e_indexado(_, _) || exp == e_puntero(_)

ambos_ok(t0, t1):
    if t0 == OK && t1 == OK then
        return OK
    else
        return ERROR
    end if

ref!(tipo):
    if tipo == t_iden(id) then
        return ref!(tipo.vinculo.tipo)
    else
        return tipo
    end if
```

4. Especificación del procesamiento de asignación de espacio.

```
var dir = 0 // primera dirección libre
var nivel = 0
var max_dir = 0 // máxima dirección ocupada

inc_dir(inc):
    dir += inc
    if dir > max_dir then
        max_dir = dir


asig_espacio(prog(bloque)):
    asig_espacio(bloque)

asig_espacio(bloque(decs, intrs)):
    dir_ant = dir
    asig_espacio(decs)
    asig_espacio(intrs)
    dir = dir_ant


asig_espacio(si_decs(decs)):
    asig_espacio1(decs)
    asig_espacio2(decs)

asig_espacio(no_decs()):
    noop

asig_espacio1(mas_decs(decs, dec)):
    asig_espacio1(decs)
    asig_espacio1(dec)

asig_espacio2(mas_decs(decs, dec)):
    asig_espacio2(decs)
    asig_espacio2(dec)

asig_espacio1(una_dec(dec)):
    asig_espacio1(dec)

asig_espacio2(una_dec(dec)):
    asig_espacio2(dec)

asig_espacio1(dec_var(tipo, id)):
```

```
    asig_tam1(tipo)
    $.dir = dir
    $.nivel = nivel
    inc_dir(tipo.tam) //var se actualiza a la siguiente dirección libre
y si se pasa de max_dir se actualiza tmb


asig_espacio2(dec_var(tipo, id)):
    asig_tam2(tipo)


asig_espacio1(dec_tipo(tipo, id)):
    asig_tam1(tipo)

asig_espacio2(dec_tipo(tipo, id)):
    asig_tam2(tipo)

asig_espacio1(dec_proc(id, pforms, bloque)):
    dir_ant = dir       // guardamos la direccion de inicio y la
max_dir
    max_dir_ant = max_dir
    nivel += 1
    $.nivel = nivel
    dir = 0            // ambos se fijan a 0 porque las direcciones en
el ámbito de un procedimiento son relativas a su registro de activación
    max_dir = 0
    asig_espacio(pforms)
    asig_espacio(bloque)
    $.tam = max_dir
    dir = dir_ant
    max_dir = max_dir_ant
    nivel -= 1

asig_espacio2(dec_proc(id, pforms, bloque)):
    noop

asig_espacio(pforms(si_pforms(pforms))):
    asig_espacio(pforms)

asig_espacio(pforms(no_pforms())):
    noop

asig_espacio(mas_pforms(pforms, pform)):
```

```
    asig_espacio(pforms)
    asig_espacio(pform)


asig_espacio(una_pform(pform)):
    asig_espacio(pform)

asig_espacio(pform(tipo, ref, id)):
    asig_espacio(tipo)
    $.dir = dir
    $.nivel = nivel
    if (ref == si_ref())
        inc_dir(1)
    else
        inc_dir(tipo.tam)


asig_espacio1(t_iden(id)):
    let $.vinculo = dec_tipo(T,id) in
        $.tam = T.tam

asig_espacio2(t_iden(id)):
    noop

asig_espacio1(t_string()):
    $.tam = 1

asig_espacio2(t_string()):
    noop

asig_espacio1(t_int()):
    $.tam = 1

asig_espacio2(t_int()):
    noop

asig_espacio1(t_bool()):
    $.tam = 1

asig_espacio2(t_bool()):
    noop

asig_espacio1(t_real()):
```

```
    $.tam = 1

asig_espacio2(t_real()):
    noop

asig_espacio1(t_array(tipo, ent)):
    asig_espacio1(tipo)
    $.tam = tipo.tam * ent

asig_espacio2(t_array(tipo, ent)):
    asig_espacio2(tipo)

asig_espacio1(t_puntero(tipo)):
    if tipo != ref(_) then
        asig_espacio1(tipo)
    end if
    $.tam = 1

asig_espacio2(t_puntero(tipo)):
    if tipo == ref(_) then
        let tipo.vinculo = dec_tipo(T, _) in
            tipo.tam = T.tam
    else
        asig_espacio2(tipo)
    end if

asig_espacio1(t_struct(camposs)):
    dirAux = dir
    dir = 0
    asig_espacio1(camposs)
    $.tam = dir
    dir = dirAux

asig_espacio2(t_struct(camposs)):
    asig_espacio2(camposs)

asig_espacio1(mas_cmp_s(camposs, campos)):
    asig_espacio1(camposs)
    asig_espacio1(campos)

asig_espacio2(mas_cmp_s(camposs, campos)):
    asig_espacio2(camposs)
    asig_espacio2(campos)
```

```
asig_espacio1(un_cmp_s(campos)):
    asig_espacio1(campos)

asig_espacio2(un_cmp_s(campos)):
    asig_espacio2(campos)

asig_espacio1(cmp_s(tipo, id)):
    $.dir = dir
    asig_espacio1(tipo)
    $.tam = tipo.tam // Mirar si el tamaño solo lo tienen los tipos y
simplemente hay que hacer += a la dir
    dir += $.tam

asig_espacio2(cmp_s(tipo, id)):
    asig_espacio2(tipo)

asig_espacio(si_intrs(intrs)):
    asig_espacio(intrs)

asig_espacio(no_intrs()):
    noop

asig_espacio(mas_intrs(intrs, intr)):
    asig_espacio(intrs)
    asig_espacio(intr)

asig_espacio(una_intr(intr)):
    asig_espacio(intr)

asig_espacio(i_eval(exp)):
    noop // no hace asig_espacio(exp)

asig_espacio(i_if(exp, bloque, ielse)):
    // no hace asig_espacio(exp)
    asig_espacio(bloque)
    asig_espacio(ielse)

asig_espacio(i_while(exp, bloque)):
    // no hace asig_espacio(exp)
    asig_espacio(bloque)

asig_espacio(i_read(exp)):
```

```
    noop // no hace asig_espacio(exp)


asig_espacio(i_write(exp)):
    noop // no hace asig_espacio(exp)

asig_espacio(i_nl()):
    noop

asig_espacio(i_new(exp)):
    noop // no hace asig_espacio(exp)

asig_espacio(i_delete(exp)):
    noop // no hace asig_espacio(exp)

asig_espacio(i_call(id, preals)):
    noop // no hace asig_espacio(preals)

asig_espacio(i_bloque(bloque)):
    asig_espacio(bloque)

asig_espacio(si_else(bloque)):
    asig_espacio(bloque)

asig_espacio(no_else()):
    noop

// no aplica    (se llamaban a hacer asignacion de espacio a las
expresiones, no les hace falta, ya que
//              ya estará definido su espacio, trabajan con la pila, no
crean nuevas variables ni describen nuevos tipos)
si_preals(preals)
no_preals()
mas_preals(preals, exp)
un_preals(exp)
e_asig(opnd0, opnd1)
e_comp(opnd0, opnd1)
e_dist(opnd0, opnd1)
e_lt(opnd0, opnd1)
e_gt(opnd0, opnd1)
e_leq(opnd0, opnd1)
e_geq(opnd0, opnd1)
e_suma(opnd0, opnd1)
```

```
e_resta(opnd0, opnd1)
e_and(opnd0, opnd1)
e_or(opnd0, opnd1)
e_mul(opnd0, opnd1)
e_div(opnd0, opnd1)
e_porcentaje(opnd0, opnd1)
e_negativo(opnd)
e_negado(opnd)
e_indexado(opnd0, opnd1)
e_campo(opnd, id)
e_puntero(opnd)
e_lit_ent(num)
e_lit_real(num)
e_true()
e_false()
e_string(string)
e_iden(id)
e_null()
```

5. Descripción del repertorio de instrucciones de la máquina-p

```
-----repertorio instrucciones-----

apila-int(n) -> apila en la cima el número n
apila-real(n) -> apila en la cima el número n
apila-bool(b) -> apila en la cima el booleano b
apila-string(s) -> apila en la cima la cadena s
apila-dir(d) -> apila en la cima el contenido de la dirección d
desapila-dir(d) -> desapila la cima y lo guarda en la dirección d

-- Tratar con objetos compuestos --
apila-ind -> desapila la cima, la interpreta como dirección y apila el
contenido de esa dirección
desapila-ind -> desapila la cima (lo interpreta como un valor),
desapila la subdima (lo interpreta como direccion) y guarda el valor en
la dirección
copia(n) -> se desapila la cima y la subcima, se interpreta como un
rango de direcciones y se copia lo que haya entre ellas n veces en el
"array" de la memoria

-- Tratar con estructuras de control while, if.. --
ir-a(d) -> salto incondicional (se salta a la dirección d)
```

```
ir-v(d) -> salto condicional (desapila la cima, si es booleano y es
cierto la ejecución sigue a la instr d, si no es cierto se sigue la
ejecución normal)
ir-f(d) -> salto condicional (desapila la cima, si es booleano y es
falso la ejecución sigue a la instr d, si es cierto se sigue la
ejecución normal)


-- Tratar con memoria dinamica --
alloc(n) ->  Reserva n celdas consecutivas en la memoria dinámica, y
apila la dirección de comienzo en
la pila de evaluación

dealloc(n) -> Desapila una dirección d de la pila de evaluación, y
considera como libres n celdas
consecutivas en la memoria dinámica, que comienzan en la dirección d.


-- Tratar con registros de activación (llamadas a procedimientos) --

activa(n,t,d) -> Reserva espacio en el segmento de pila de registros de
activación
    para ejecutar un procedimiento que tiene nivel de anidamiento n y
tamaño de datos
    locales t. Así mismo, almacena en la zona de control de dicho
registro d como
    dirección de retorno. También almacena en dicha zona de control el
valor del display
    de nivel n. Por último, apila en la pila de evaluación la dirección
de comienzo de los
    datos en el registro creado.

apilad(n) -> Apila en la pila de evaluación el valor del display de
nivel n

desapilad(n) -> Desapila una dirección d de la pila de evaluación en el
display de nivel n.

desactiva(n,t) -> Libera el espacio ocupado por el registro de
activación actual,
    restaurando adecuadamente el estado de la máquina. n indica el
nivel de anidamiento
```

```
    del procedimiento asociado; t el tamaño de los datos locales. De
esta forma, la
    instrucción: (i) apila en la pila de evaluación la dirección de
retorno; (ii) restaura el
    valor del display de nivel n al antiguo valor guardado en el
registro; (iii) decrementa el
    puntero de pila de registros de activación en el tamaño ocupado por
el registro.

dup -> Consulta el valor v de la cima de la pila de evaluación, y apila
de nuevo dicho valor (es decir, duplica la cima de la pila de
evaluación)

ir-ind -> Desapila una dirección d de la pila de evaluación, y realiza
un salto incondicional a dicha dirección.

stop ->Detiene la máquina.

suma -> desapila la cima y la subcima, los suma y lo apila
resta -> desapila la cima y la subcima, los resta y lo apila
mul -> desapila la cima y la subcima, los multiplica y lo apila
div -> desapila la cima y la subcima, los divide y lo apila
mod -> desapila la cima y la subcima, reliza la operacion del modulo y
apila el resultado
or -> desapila la cima y la subcima, realiza la operacion or logica y
apila el resultado
and -> desapila la cima y la subcima, realiza la operacion and logica y
apila el resultado
menor -> desapila la cima y la subcima, realiza la comparacion menor
que y apila el resultado
mayor -> desapila la cima y la subcima, realiza la comparacion mayor
que y apila el resultado
menor_igual -> desapila la cima y la subcima, realiza la comparacion
menor o igual que y apila el resultado
mayor_igual -> desapila la cima y la subcima, realiza la comparacion
mayor o igual que y apila el resultado
igual -> desapila la cima y la subcima, calcula si ambos son iguales y
apila el resultado
dist -> desapila la cima y la subcima, calcula si ambos son distintos y
apila el resultado
neg -> desapila la cima, invierte el valor y lo apila // Se podria
implementar como un apila_int(0) y un resta()
```

```
not -> desapila la cima, niega el valor y lo apila // Se podria
implementar con un apila_bool()

write -> desapila la cima y la escribe por la salida
read -> le la entrada y lo apila

int2real -> desapila la cima, si es un entero lo pasa a real y lo
apila, sino da fallo
```

6. Especificación del procesamiento de etiquetado.

```
var sub_pendientes=pila_vacia()
var etq_final

etiquetado(prog(bloque)):
    $.prim = etq_final
    etiquetado(bloque)
    etq_final++
    while ¬ es_vacia(sub_pendientes)
        sub = cima(sub_pendientes)
        desapila(sub_pendientes)
        let sub = dec_proc(id, pforms, bloque2) in
            sub.prim = etq_final
            etq_final++
            recolecta_subs(pforms)
            etiquetado(bloque2)
            etq_final += 2
            sub.sig = etq_final
        end let
    end while
    $.sig = etq_final

etiquetado(bloque(decs, intrs)):
    $.prim = etq_final
    recolecta_subs(decs)
    etiquetado(intrs)
    $.sig = etq_final

recolecta_subs(si_decs(decs)):
    recolecta_subs(decs)

recolecta_subs(no_decs()):
    noop
```

```
recolecta_subs(mas_decs(decs, dec)):
    recolecta_subs(decs)
    recolecta_subs(dec)

recolecta_subs(una_dec(dec)):
    recolecta_subs(dec)

recolecta_subs(dec_var(tipo, id)):
    noop

recolecta_subs(dec_tipo(tipo, id)):
    noop

recolecta_subs(dec_proc(id, pforms, bloque)):
    apila(sub_pendientes, $)

etiquetado(si_intrs(intrs)):
    etiquetado(intrs)

etiquetado(no_intrs()):
    noop

etiquetado(mas_intrs(intrs,intr)):
    etiquetado(intrs)
    etiquetado(intr)

etiquetado(una_intr(intr)):
    etiquetado(intr)

etiquetado(i_eval(exp)):
    etiquetado(exp)
    etq_final++

etiquetado_acc_valor(exp):
    if es_designador(exp) then
        etq_final++
    end if

etiquetado_exp(exp):
    etiquetado(exp)
    etiquetado_acc_valor(exp)
```

```
etiquetado(i_if(exp,bloque,i_else)):
    $.prim = etq_final
    etiquetado_exp(exp)
    etq_final++
    etiquetado(bloque)
    if i_else == si_else() then
        etq_final++
    end if
    $.sig = etq_final
    etiquetado(i_else)
    $.fin = etq_final

etiquetado(i_while(exp,bloque)):
    $.prim = etq_final
    etiquetado_exp(exp)
    etq_final++
    etiquetado(bloque)
    etq_final++
    $.sig = etq_final

etiquetado(i_read(exp)):
    etiquetado(exp)
    etq_final += 2

etiquetado(i_write(exp)):
    etiquetado_exp(exp)
    etq_final += 1

etiquetado(i_nl()):
    etq_final += 1

etiquetado(i_new(exp)):
    etiquetado(exp)
    etq_final += 2

etiquetado(i_delete(exp)):
    etiquetado(exp)
    etq_final += 2

etiquetado(i_call(id, preals)):
    etq_final++
    etiquetado_paso_param($.vinculo.pforms, preals)
    etq_final++
```

```
etiquetado(i_prog(bloque)):
    etiquetado(bloque)

etiquetado(si_else(bloque)):
    etiquetado(bloque)
    $.fin = etq_final

etiquetado(no_else()):
    noop

etiquetado(e_asig(opnd0, opnd1)):
    etiquetado(opnd0)
    etiquetado(opnd1)
    if opnd0.tipo == t_real() and opnd1.tipo == t_int():
        etiquetado_acc_valor(opnd1)
        etq_final++
    end if
    etq_final++

etiquetado(e_comp(opnd0, opnd1)):
    etiquetado_opnds(opnd1,opnd2)

etiquetado(e_dist(opnd0, opnd1)):
    etiquetado_opnds(opnd1,opnd2)

etiquetado(e_lt(opnd0, opnd1)):
    etiquetado_opnds(opnd1,opnd2)

etiquetado(e_gt(opnd0, opnd1)):
    etiquetado_opnds(opnd1,opnd2)

etiquetado(e_leq(opnd0, opnd1)):
    etiquetado_opnds(opnd1,opnd2)

etiquetado(e_geq(opnd0, opnd1)):
    etiquetado_opnds(opnd1,opnd2)

etiquetado(e_suma(opnd0, opnd1)):
    etiquetado_opnds(opnd1,opnd2)

etiquetado(e_resta(opnd0, opnd1)):
```

```
    etiquetado_opnds(opnd1,opnd2)

etiquetado(e_and(opnd0, opnd1)):
    etiquetado_opnds(opnd1,opnd2)

etiquetado(e_or(opnd0, opnd1)):
    etiquetado_opnds(opnd1,opnd2)

etiquetado(e_mul(opnd0, opnd1)):
    etiquetado_opnds(opnd1,opnd2)

etiquetado(e_div(opnd0, opnd1)):
    etiquetado_opnds(opnd1,opnd2)

etiquetado(e_porcentaje(opnd0, opnd1)):
    etiquetado_opnds(opnd1,opnd2)

etiquetado(e_negativo(opnd)):
    etiquetado_opnd(opnd)

etiquetado(e_negado(opnd)):
    etiquetado_opnd(opnd)

etiquetado(e_indexado(opnd0, opnd1)):
    etiquetado(opnd0)
    etiquetado_exp(opnd1)
    etq_final += 3

etiquetado(e_campo(opnd, id)):
    etiquetado(opnd)
    etq_final+=2

etiquetado(e_puntero(opnd)):
    etiquetado_exp(opnd)
    etq_final++

etiquetado(e_lit_ent(num)):
    etq_final++

etiquetado(e_lit_real(num)):
    etq_final++

etiquetado(e_true()):
```

```
    etq_final++

etiquetado(e_false()):
    etq_final++

etiquetado(e_string(string)):
    etq_final++

etiquetado(e_iden(id)):
    etiquetado_acc_id(id)

etiquetado(e_null()):
    etq_final++

//Funciones auxiliares

etiquetado_acc_id(dec_var(tipo, id)):
    if $.nivel == 0 then
        etq_final++
    else
        etq_final += 3
    end if

etiquetado_acc_id(pform(tipo, ref, id)):
    etq_final += 3
    if ref == si_ref() then:
        etq_final++
    end if

etiquetado_paso_param(proc(id, pforms, bloque), exp):
    etq_final+= 3
    etiquetado(exp)
    if ref == no_ref() and tipo == t_real() and exp.tipo == t_int():
        etq_final+=2
    end if
    etq_final++

etiquetado_opnds(opnd1,opnd2):
    etiquetado_exp(opnd0)
    if opnd0.tipo == t_int() and opnd1.tipo == t_real() then:
        etq_final++
    end if
```

```
    etiquetado_exp(opnd1)
    if opnd0.tipo == t_real() and opnd1.tipo == t_int() then:
        etq_final++
    end if

    etq_final++

etiquetado_opnd(exp):
    etiquetado_exp(exp)
    etq_final++

etiquetado_paso_param(no_pforms(), no_preals()):
    noop

etiquetado_paso_param(si_pforms(pforms), si_preals(preals)):
    etiquetado_paso_param(pforms, preals)

etiquetado_paso_param(mas_pforms(pforms, pform), mas_preals(preals,
exp)):
    etiquetado_paso_param(pforms, preals)
    etiquetado_paso_param(pform, exp)

etiquetado_paso_param(una_pform(pform), un_preals(exp)):
    etiquetado_paso_param(pform, exp)
```

7. Especificación del procesamiento de generación de código

```
var sub_pendientes = pila_vacia()

gen_code(prog(bloque))
    gen_code(bloque)
    emit stop()

    while not es_vacia(sub_pendientes)
        sub = cima(sub_pendientes)
        desapila(sub_pendientes)
        let sub = dec_proc(id, param, decs, Is) in
            emit desapilad(sub.nivel)
            recolecta_subs(Decs)
            gen_code(Is)
            emit desactiva(sub.nivel, sub.tam)
            emit ir_ind()
        end let
```

```
    end while

gen_code(bloque(decs, intrs)):
    recolecta_subs(decs)
    gen_code(intrs)


recolecta_subs(si_decs(decs)):
    recolecta_subs(decs)

recolecta_subs(no_decs()):
    noop

recolecta_subs(mas_decs(decs, dec)):
    recolecta_subs(decs)
    recolecta_subs(dec)

recolecta_subs(una_dec(dec)):
    recolecta_subs(dec)

recolecta_subs(dec_var(tipo, id)):
    noop

recolecta_subs(dec_tipo(tipo, id)):
    noop

recolecta_subs(dec_proc(id, pforms, bloque)):
    apila(sub_pendientes, $)

gen_code(si_intrs(intrs)):
    gen_code(intrs)

gen_code(no_intrs()):
    noop

gen_code(mas_intrs(intrs, intr)):
    gen_code(intrs)
    gen_code(intr)

gen_code(una_intr(intr)):
    gen_code(intr)

gen_code(i_eval(exp)):
```

```
    gen_code(exp)
    emit desapila()  // desechar el valor

gen_code(i_if(exp, bloque, ielse)):
    gen_code_exp(exp)

    emit ir_f($.sig) // si es falso saltamos al else o final del if en
caso de no haber else
    gen_code(bloque) //codigo del bloque if
    gen_code(ielse) // codigo del bloque else

gen_code(i_prog(bloque)):
    gen_code(bloque)

gen_code(si_else(bloque)):
    emit ir_a($.fin) // Hemos ejecutado el bloque del if, saltamos
incondicionalmente al final (no se ejecuta else)
    gen_code(bloque)

gen_code(no_else(bloque)):
    noop

gen_code(i_while(exp, bloque)):
    gen_code_exp(exp)

    emit ir_f($.sig) // si es falso saltamos a sig (direccion de la
instr que sigue al while)
    gen_code(bloque)
    emit ir_a($.prim) // siempre se vuelve a la primera instrucción del
while


gen_code(i_read(exp)):
    gen_code(exp) // dirección de lectura se deja en la cima
    emit read      // lee el valor y se guarda en la cima
    emit desapila_ind  //guarda la cima en la dirección de la subcima

gen_code(i_write(exp)):
    gen_code_exp(exp)
    emit write


gen_code(i_nl()):
```

```
    emit nl  // no sé si hay que hacer algo

gen_code(i_new(exp)):
    gen_code(exp)  //Dirección de comienzo de la expresion
    emit alloc(exp.tipo.tipo.tamaño)     //reservamos tantas celdas
como el tamaño del tipo de la expresión y apilamos la dir de comienzo
    emit desapila_ind  // en la cima tenemos el comienzo de la mem
dinamica asignada, en la subcima la dir del propio puntero, guardamos
en la subcima la cima

gen_code(i_delete(exp)):
    gen_code(exp)  // Direccion donde está el puntero
    emit apila_ind  // apilamos el valor apuntado por el puntero
    emit dealloc(exp.tipo.tipo.tamaño) // liberamos la memoria de dicho
tamaño // tipo.tipo porque es tipo puntero

gen_code_exp(exp):
    gen_code(exp)
    gen_acc_valor(exp)

gen_code(e_asig(opnd0, opnd1)):
   gen_code(opnd0)
   gen_code(opnd1)

    if opnd0.tipo == t_real() and opnd1.tipo == t_int():
        gen_acc_valor(opnd1)

        emit int2real //convertimos el valor a real
        emit desapila_ind //guardamos el valor convertido en la
dirección de opnd0
    else:
        if es_designador(opnd1):
            emit copia(tamaño(opnd1.tipo)) // se va copiando celda a
celda el valor
        else:
            emit desapila_ind // guarda en la dirección de opnd0 el
valor de la cima (opnd1)

gen_code_bin(opnd0, opnd1):
    gen_code_exp(opnd0)
    if opnd0.tipo == t_int() and opnd1.tipo == t_real():
        emit int2real
```

```
    gen_code_exp(opnd1)
    if opnd0.tipo == t_real() and opnd1.tipo == t_int():
        emit int2real

gen_code(e_comp(opnd0, opnd1)):
    gen_code_bin(opnd0, opnd1)
    emit comp

gen_code(e_dist(opnd0, opnd1)):
    gen_code_bin(opnd0, opnd1)
    emit dist

gen_code(e_lt(opnd0, opnd1)):
    gen_code_bin(opnd0, opnd1)
    emit lt

gen_code(e_gt(opnd0, opnd1)):
    gen_code_bin(opnd0, opnd1)
    emit gt

gen_code(e_leq(opnd0, opnd1)):
    gen_code_bin(opnd0, opnd1)
    emit leq

gen_code(e_geq(opnd0, opnd1)):
    gen_code_bin(opnd0, opnd1)
    emit geq

gen_code(e_suma(opnd0, opnd1)):
    gen_code_bin(opnd0, opnd1)
    emit suma

gen_code(e_resta(opnd0, opnd1)):
    gen_code_bin(opnd0, opnd1)
    emit resta

gen_code(e_mul(opnd0, opnd1)):
    gen_code_bin(opnd0, opnd1)
    emit mul

gen_code(e_div(opnd0, opnd1)):
    gen_code_bin(opnd0, opnd1)
    emit div
```

```
gen_code(e_porcentaje(opnd0, opnd1)):
    gen_code_bin(opnd0, opnd1)
    emit mod

gen_code(e_and(opnd0, opnd1)):
    gen_code_bin(opnd0, opnd1)
    emit and

gen_code(e_or(opnd0, opnd1)):
    gen_code_bin(opnd0, opnd1)
    emit or

gen_code(e_negativo(opnd)):
    gen_code_exp(opnd)
    emit neg //negamos el valor de la cima

gen_code(e_negado(opnd)):
    gen_code_exp(opnd)
    emit not //negamos el valor de la cima

gen_code(i_call(id, preals)):
    proc = $.vinculo
    //preparar el resgistro de activación
    emit activa(proc.nivel, proc.tam, $.sig) //en la cima tenemos la
dirección de comienzo del registro de activación

    id_pform = 0

    gen_code_params(proc.pforms, preals)
    emit ir_a(proc.prim)

gen_code_params(no_pforms(), no_preals()):
    noop

gen_code_params(si_pforms(pforms), si_preals(preals)):
    gen_code_params(pforms, preals)

gen_code_params(mas_pforms(pforms, pform), mas_preals(preals, exp)):
    gen_code_params(pforms, preals)
    gen_code_params(pform, exp)

gen_code_params(una_pform(pform), un_preals(exp)):
```

```
    gen_code_params(pform, exp)

gen_code_params(pform(tipo, ref, id), exp):
    emit dup //duplicamos la cima (dir de comienzo de los datos del
resgistro de activación)
    emit apila_int(pform.dir) //apilamos el tamaño del parámetro
    emit suma //tenemos en la cima la dirección de comienzo del
parámetro
    gen_code(exp) //apilamos el valor/dir de comienzo del parámetro

    if ref == no_ref():
        if tipo == t_real() and exp.tipo == t_int():
            gen_acc_valor(exp)
            emit int2real //convertimos el valor a real
            emit desapila_ind // cima es el valor y subcima la
dirección de comienzo del parámetro guardamos el valor en la dirección
del param
        else if es_designador(exp): // param formal por valor y real es
designador
            emit copia(tipo.tamaño) //cima (dir apuntada preal) subcima
(dir inicio del param en el registro activacion) se realiza copia del
tamaño del tipo
        else:
            emit desapila_ind // cima es el valor y subcima la
dirección de comienzo del parámetro guardamos el valor en la dirección
del param
    else:
        emit desapila_ind // como ambos son punteros, en cima tenemos
el valor apuntado por el preal, que se guarda en la dirección del pform


gen_code(i_bloque(bloque)):
    gen_code(bloque)

gen_code(bloque(decs, intrs)):
    recolecta_subs(decs)
    gen_code(intrs)

gen_code(e_indexado(opnd0, opnd1)):
    gen_code(opnd0) // obtenemos la dirección del array
    gen_code(opnd1) // obtenemos el índice
    gen_acc_valor(opnd1)
```

```
    emit apila_int(opnd0.tipo.tipo.tamaño) //apilamos el tamaño del
tipo del array
    emit mul //apilamos el desplazamiento del índice
    emit suma // a la dirección de comenzo del array le sumamos el
desplazamiento del índice

gen_code(e_campo(opnd, id)):
    gen_code(opnd) //determinamos la dirección de E
    d = desplazamiento(opnd.tipo, id) //obtenemos el desplazamiento del
campo id, se calcula en asignacion de espacio
    emit apila_int(d)
    emit suma // desplazamiento del campo + dirección de comienzo de E

gen_code(e_puntero(opnd)):
    gen_code(opnd) // determinamos dirección de E
    emit apila_ind //apilamos la dirección a la que apunta el puntero

gen_code(e_lit_ent(num)):
    emit apila_int(num) //apilamos el literal entero

gen_code(e_lit_real(num)):
    emit apila_real(num) //apilamos el literal real

gen_code(e_true()):
    emit apila_bool(true) //apilamos el literal booleano true

gen_code(e_false()):
    emit apila_bool(false) //apilamos el literal booleano false

gen_code(e_string(string))
    emit apila_string(string) //apilamos el literal string

gen_code(e_null()):
    emit apila_int(-1) //apilamos el literal entero _1 (null)

gen_code(e_iden(id)):
    gen_acc_id($.vinculo)

gen_acc_id(dec_var(tipo, id)):
    if $.nivel == 0: // variable global
        emit apila_int($.vinculo.dir) //apilamos la dirección de la
variable
    else:
```

```
        emit apilad($.nivel) //apilamos el display de nivel $.nivel
        emit apila_int($.vinculo.dir) //apilamos la dirección de la
variable
        emit suma //apilamos la dirección de la variable (display + dir
var)
    end if

gen_acc_id(pform(tipo, ref, id)):
    emit apilad($.nivel) //apilamos el display de nivel $.nivel
    emit apila_int($.vinculo.dir) //apilamos la dirección de la
variable
    emit suma //apilamos la dirección de la variable (display + dir
var)
    if ref == si_ref() then:
        emit apila_ind()
    end if

gen_acc_valor(exp):
    if es_designador(exp):
        emit apila_ind //apilamos el valor apuntado por el designador


desplazamiento(t_struct(camposs), idCampo):
    return desplazamiento(camposs, idCampo)

desplazamiento(mas_cmp_s(camposs, cmp_s(tipo, id)), idCampo):
    if idCampo == id then:
        return cmp_s.dir
    else
        return desplazamiento(camposs, idCampo)
    end if

desplazamiento(un_cmp_s(cmp_s(tipo, id)), idCampo):
    if idCampo == id then:
        return cmp_s.dir
    else
        return error // No deberia de ocurrir
    end if
```