PRÁCTICA PL: CUARTA FASE

Tiny(1)

Integrantes:

David Davó Laviña

Ela Katherine Shepherd Arévalo Grupo 12

PASO 0: Especificación sintaxis abstracta

prog_sin_decs: Insts → Prog
prog_con_decs: Decs × Insts → Prog
dec_una: Dec → Decs
decs_muchas: Decs × Dec → Decs
var: Tipo × string → Dec
type: Tipo × string → Dec
proc: string × PFs × Bloque → Dec
param_f_sin: → PFs
param_f_con_una: PF → PFs
param_f_con_muchas: PFs × PF → PFs
param_f_ref: Tipo × string → PF
param_f_noref: Tipo × string → PF
tipo_array: String × Tipo → Tipo
tipo_record: Campos → Tipo
tipo_pointer: Tipo → Tipo
tipo_iden: string → Tipo
tipo_int: → Tipo
tipo_real: → Tipo
tipo_bool: → Tipo
tipo_string: → Tipo
campos_uno: Campo → Campos
campos_muchos: Campos × Campo→Campos
campo: Tipo × string → Campo
inst_una: Inst → Insts
insts_muchas: Insts × Inst → Insts
e_igual: Exp × Exp →Inst
if: Exp × PInst →Inst
ifelse: Exp × PInst × PInst→Inst
while: Exp × PInst →Inst
read: Exp →Inst
write: Exp →Inst
nl: →Inst
new: Exp →Inst
delete: Exp →Inst
call: string × PR →Inst
bl: Bloque →Inst
lista_sin: →PInst
lista_con: Insts →PInst
param_r_sin: → PR
param_r_con_una: Exp → PR

param_r_con_muchas: PR × Exp→ PR
bloque_sin: → Bloque
bloque_con: Prog → Bloque
entero: string → Exp
real: string → Exp
cadena: string → Exp
verdadero: → Exp
falso: → Exp
null: → Exp
identificador: string → Exp
suma: Exp × Exp → Exp
resta: Exp × Exp → Exp
and: Exp × Exp → Exp
or: Exp × Exp → Exp
menor: Exp × Exp → Exp
men_ig: Exp × Exp → Exp
mayor: Exp × Exp → Exp
may_ig: Exp × Exp → Exp
igual: Exp × Exp → Exp
desigual: Exp × Exp → Exp
mul: Exp × Exp → Exp
div: Exp × Exp → Exp
modulo: Exp × Exp → Exp
m_unario: Exp → Exp
not: Exp → Exp
indexacion: Exp × Exp → Exp
acc_registro: Exp × string→ Exp
acc_registro_ind: Exp × string→Exp
indireccion: Exp → Exp

PASO 1: Procesamiento de vinculación

Asociamos cada uso de un identificador con su declaración.

> **Nota:** `Tabla_sim` es una estructura de datos que contiene una pila de las tablas de símbolos de los distintos niveles. Sus métodos son:
>
> - anida: Crea una nueva tabla de símbolos y la añade arriba en la pila
> - desanida: Quita la cima de la pila
> - contieneAny(id): Comprueba si el identificador está en alguna tabla de símbolos de la pila
> - valorPara(id): Obtiene el elemento asociado al identificador de alguna de las tablas de símbolos de la pila
> - add(id,elem): Añade un elemento a la tabla de símbolos
> - contieneAct(id): Comprueba si el identificador está en la tabla de símbolos de la pila (usado al definir un nuevo identificador, pues pueden sobreescribirse).
>
> En general, antes de añadir a la tabla de simbolos se usa contieneAct, pero cuando vamos a vincular usamos contieneAny.

```
global (TablaSim) tabla_sim;

vincula(prog_sin_decs(Insts)) {
	vincula(Insts);
}

vincula(prog_con_decs(Decs, Insts)){
	construye(Decs); //crea una tabla de símbolos (si bloque_actual es mayor que 0, además, se crea un puntero a la tabla de símbolos de identificador menor)
	vinculaPointer(Decs); // Realizamos una segunda pasada para vincular los pointer. Al final no se ha terminado de desarrollar el procesamiento de vinculaPointer
	vincula(Insts);
}

construye(dec_una(Dec)){
	construye(Dec);
}

construye(decs_muchas(Decs, Dec)){
	construye(Decs);
	construye(Dec);
}

construye(var(Tipo, iden)){
```

```
      if (tabla_sim.contieneAct(id)) error //comprueba si existe un
campo con ese nombre en la tabla de símbolos hijo Y YA. No se mira en
la tabla padre
      else {
            tabla_sim.add(iden, $); //se añade en la tabla hijo
      }
      vincula(Tipo)
}

construye(type(Tipo, iden)){
      if (tabla_sim.contieneAct(iden)) error
      else {
            tabla_sim.add(iden, Tipo);
      }
}

construye(proc(iden, PF, Bloque)){
      if (tabla_sim.contieneAct(iden)) error
      else {
            tabla_sim.add(iden, $);
            tabla_sim.anida();
            construye(PF);
            vincula(Bloque);
            tabla_sim.desanida();
      }
}

construye(param_f_sin()) {
      // No hacemos nada
}

construye(param_f_con_una(PF)){
      construye(PF);
}

construye(param_f_con_muchas(PFS, PF)){
      construye(PFS);
      construye(PF);
}

construye(param_f_ref(Tipo, iden)){
      if (tabla_sim.contieneAct(iden)) error
      else {
            tabla_sim.add(iden, $);
      }
}
```

```
construye(param_f_noref(Tipo, iden)){
      if (tabla_sim.contieneAct(iden)) error
      else {
            tabla_sim.add(iden, $);
      }
}

vincula(tipo_array(n,T)) {
      vincula(T)
}

vincula(tipo_record(Cs)) {
      vincula(Cs)
}

vincula(tipo_pointer(T)) {
      // Ya se hará en vinculapointer
}

vincula(tipo_iden(Id)) {
      if (!tabla_sim.contieneAny(iden)) error
      else $.vinculo = tabla_sim.obtiene(iden)
}

vincula(insts_muchas(Insts, Inst)){
      vincula(Insts);
      vincula(Inst);
}

vincula(inst_una(Inst)){
      vincula(Inst);
}

vincula(e_igual(E0, E1)){
      vincula(E0);
      vincula(E1);
}

vincula(if(E, PI)){
      vincula(E);
      vincula(PI);
}

vincula(ifelse(E, PI0, PI1)){
      vincula(E);
```

```
	vincula(PI0);
	vincula(PI1);
}

vincula(while(E, PI)){
	vincula(E);
	vincula(PI);
}

vincula(read(E)){
	vincula(E);
}


vincula(write(E)){
	vincula(E);
}

vincula(new(E)){
	vincula(E);
}

vincula(delete(E)){
	vincula(E);
}

vincula(call(iden, PR)){
	if (!tabla_sim.contieneAny(iden)) error //para comprobar si
está en la tabla de símbolos, mira la tabla de símbolos actual, si no
está, en la tabla padre, y así...
	else{
		$.vinculo = tabla_sim.valorPara(iden);
		vincula(PR);
	}}
vincula(bl(Bloque)){
	tabla_sim.anida();
	vincula(Bloque);
	tabla_sim.desanida();
}

vincula(lista_con(Insts)){
	vincula(Insts);
}

vincula(param_r_con_una(E)){
	vincula(E);
```

```
}

vincula(param_r_con_muchas(PR, E)){
      vincula(PR);
      vincula(E);
}

vincula(bloque_con(Prog)){
      vincula(Prog);
}

vincula(identificador(iden)){
      if (!tabla_sim.contieneAny(iden)) error
      else{
            $.vinculo = tabla_sim.valorPara(iden);
      }
}

vincula(suma(E0, E1)){
      vincula(E0);
      vincula(E1);
}

vincula(resta(E0, E1)){
      vincula(E0);
      vincula(E1);
}
vincula(and(E0, E1)){
      vincula(E0);
      vincula(E1);
}

vincula(or(E0, E1)){
      vincula(E0);
      vincula(E1);
}

vincula(menor(E0, E1)){
      vincula(E0);
      vincula(E1);
}

vincula(men_ig(E0, E1)){
      vincula(E0);
      vincula(E1);
}
```

```
vincula(mayor(E0, E1)){
      vincula(E0);
      vincula(E1);
}

vincula(may_ig(E0, E1)){
      vincula(E0);
      vincula(E1);
}

vincula(igual(E0, E1)){
      vincula(E0);
      vincula(E1);
}

vincula(desigual(E0, E1)){
      vincula(E0);
      vincula(E1);
}


vincula(mul(E0, E1)){
      vincula(E0);
      vincula(E1);
}

vincula(div(E0, E1)){
      vincula(E0);
      vincula(E1);
}

vincula(modulo(E0, E1)){
      vincula(E0);
      vincula(E1);
}

vincula(m_unario(E)){
      vincula(E);
}

vincula(not(E)){
      vincula(E);
}

vincula(indexacion(E0, E1)){
```

```
      vincula(E0);
      vincula(E1);
}

vincula(acc_registro(E, iden)){
      vincula(E);
}

vincula(acc_registro_ind(E, iden)){
      vincula(E);
}

vincula(indireccion(E)){
      vincula(E);
}
```

PASO 2: Procesamiento de comprobación de tipos

```
comprueba_tipos(prog_sin_decs(Insts)){
      comprueba_tipos(Insts);
      $.tipo = Insts.tipo;
}

comprueba_tipos(prog_con_decs(Decs, Insts)){
      comprueba_tipos(Insts);
      $.tipo = Insts.tipo;
}

comprueba_tipos(inst_una(Inst)){
      comprueba_tipos(Inst);
      $.tipo = Inst.tipo;
}

comprueba_tipos(insts_muchas(Insts, Inst)){
      comprueba_tipos(Insts);
      comprueba_tipos(Inst);
      if (Insts.tipo == ok && Inst.tipo == ok){
            $.tipo = ok;
      }
```

```
	else $.tipo = error;
}

comprueba_tipos(e_igual(E0, E1)){
	comprueba_tipos(E0);
	comprueba_tipos(E1);
	if (E0.esDesignador() && compatibilidad(E0.tipo,
E1.tipo)){//esDesignador comprueba si una expresión es designador
(identificador, puntero, indexación, acceso a registro)
		$.tipo = ok;
	}
	else $.tipo = error;
}

comprueba_tipos(if(E, PI)){
	comprueba_tipos(E);
	comprueba_tipos(PI);
	if (E.tipo == bool && PI.tipo == ok){
		$.tipo = ok;
	}
	else $.tipo = error;
}

comprueba_tipos(ifelse(E, PI0, PI1)){
	comprueba_tipos(E);
	comprueba_tipos(PI0);
	comprueba_tipos(PI1);
	if (E.tipo == bool && PI0.tipo == ok && PI1.tipo == ok){
		$.tipo = ok;
	}
	else $.tipo = error;
}

comprueba_tipos(while(E, PI)){
	comprueba_tipos(E);
	comprueba_tipos(PI);
	if (E.tipo == bool && PI.tipo == ok){
		$.tipo = ok;
	}
	else $.tipo = error;
}

comprueba_tipos(read(E)){
	comprueba_tipos(E);
	if (E.esDesignador() && (E.tipo == int | E.tipo == real |
E.tipo == string)){
```

```
            $.tipo = ok;
      }
      else $.tipo = error;
}

comprueba_tipos(write(E)){
      comprueba_tipos(E);
      if (E.tipo == int | E.tipo == real |  E.tipo == string | E.tipo
== bool){
            $.tipo = ok;
      }
      else $.tipo = error;
}

comprueba_tipos(nl()){
      $.tipo = ok;
}
```

```
comprueba_tipos(new(E)){
      comprueba_tipos(E);
      if (E.tipo == pointer){
            $.tipo = ok;
      }
      else $.tipo = error;
}

comprueba_tipos(delete(E)){
      comprueba_tipos(E);
      if (E.tipo == pointer){
            $.tipo = ok;
      }
      else $.tipo = error;
}

comprueba_tipos(call(iden, PI)){
      comprueba_tipos(PI);
      sea $.vinculo = proc(i, PF, Bloque) en{
            comprueba_tipos(PF);
            for (i = 0; i por cada parámetro real de PI + 1; i++){
                  if ((PI[i] == null && PF[i] != null) || (PI[i] !=
null && PF[i] == null)){
```

```
                    $.tipo = error;
                }
                else if (PI[i].tipo != PF[i].tipo){
                    $.tipo = error;
                }
                else if (PF[i].vinculo == param_f_ref(Tipo, iden) &&
!PI[i].esDesignador()){
                    $.tipo = error;
                }
                else if (PI[i] == null && PF[i] == null){
                    $.tipo = ok;
                }
    }
}

comprueba_tipos(bl(B)){
    $.tipo = B.tipo;
}

comprueba_tipos(lista_sin()){
    $.tipo = ok;
}

comprueba_tipos(lista_con(Insts)){
    $.tipo = Insts.tipo;
}

comprueba_tipos(param_r_con_una(E)){
    comprueba_tipos(E);
    $.tipo = E.tipo;
}

comprueba_tipos(param_r_con_muchas(PR, E)){
    comprueba_tipos(PR);
    comprueba_tipos(E);
    $.tipo = E.tipo;
}

comprueba_tipos(bloque_con(Prog)){
    $.tipo = Prog.tipo;
}

comprueba_tipos(bloque_sin()){
    $.tipo = ok;
}
```

```
comprueba_tipos(entero(iden)){
      $.tipo = int;
}

comprueba_tipos(real(iden)){
      $.tipo = real;
}

comprueba_tipos(cadena(iden)){
      $.tipo = string;
}

comprueba_tipos(verdadero()){
      $.tipo = bool;
}

comprueba_tipos(falso()){
      $.tipo = bool;
}

comprueba_tipos(null()){
      $.tipo = null;
}

comprueba_tipos(identificador(iden)){
      if ($.vinculo == var(T, iden)){
            sea $.vinculo == var(T, iden) en {
                  $.tipo = T;
            }
      }
      else $.tipo = error;
}

comprueba_tipos(suma(E0, E1)){
      comprueba_tipos(E0);
      comprueba_tipos(E1);
      if (E0.tipo == int && E1.tipo == int){
            $.tipo = int;
      }
      else if ((E0.tipo == real && (E1.tipo == real || E1.tipo ==
int)) || (E1.tipo == real && (E0.tipo == real || E0.tipo == int))){
            $.tipo = real;
      }
      else $.tipo = error;
}
```

```
comprueba_tipos(resta(E0, E1)){
      comprueba_tipos(E0);
      comprueba_tipos(E1);
      if (E0.tipo == int && E1.tipo == int){
            $.tipo = int;
      }
      else if ((E0.tipo == real && (E1.tipo == real || E1.tipo ==
int)) || (E1.tipo == real && (E0.tipo == real || E0.tipo == int))){
            $.tipo = real;
      }
      else $.tipo = error;
}

comprueba_tipos(and(E0, E1)){
      comprueba_tipos(E0);
      comprueba_tipos(E1);
      if (E0.tipo == bool && E1.tipo == bool){
            $.tipo = bool;
      }
      else $.tipo = error;
}

comprueba_tipos(or(E0, E1)){
      comprueba_tipos(E0);
      comprueba_tipos(E1);
      if (E0.tipo == bool && E1.tipo == bool){
            $.tipo = bool;
      }
      else $.tipo = error;
}

comprueba_tipos(menor(E0, E1)){
      comprueba_tipos(E0);
      comprueba_tipos(E1);
      if (((E0.tipo == int || E0.tipo == real) && (E1.tipo == int ||
E1.tipo == real)) ||
      (E0.tipo == bool && E1.tipo == bool) ||
      (E0.tipo == string && E1.tipo == string)){
            $.tipo = bool;
      }
      else $.tipo = error;
}

comprueba_tipos(men_ig(E0, E1)){
      comprueba_tipos(E0);
      comprueba_tipos(E1);
```

```
      if (((E0.tipo == int || E0.tipo == real) && (E1.tipo == int ||
E1.tipo == real)) ||
      (E0.tipo == bool && E1.tipo == bool) ||
      (E0.tipo == string && E1.tipo == string)){
            $.tipo = bool;
      }
      else $.tipo = error;
}

comprueba_tipos(mayor(E0, E1)){
      comprueba_tipos(E0);
      comprueba_tipos(E1);
      if (((E0.tipo == int || E0.tipo == real) && (E1.tipo == int ||
E1.tipo == real)) ||
      (E0.tipo == bool && E1.tipo == bool) ||
      (E0.tipo == string && E1.tipo == string)){
            $.tipo = bool;
      }
      else $.tipo = error;
}

comprueba_tipos(may_ig(E0, E1)){
      comprueba_tipos(E0);
      comprueba_tipos(E1);
      if (((E0.tipo == int || E0.tipo == real) && (E1.tipo == int ||
E1.tipo == real)) ||
      (E0.tipo == bool && E1.tipo == bool) ||
      (E0.tipo == string && E1.tipo == string)){
            $.tipo = bool;
      }
      else $.tipo = error;
}

comprueba_tipos(igual(E0, E1)){
      comprueba_tipos(E0);
      comprueba_tipos(E1);
      if (((E0.tipo == int || E0.tipo == real) && (E1.tipo == int ||
E1.tipo == real)) ||
      (E0.tipo == bool && E1.tipo == bool) ||
      (E0.tipo == string && E1.tipo == string) ||
      ((E0.tipo == pointer || E0.tipo == null) && (E1.tipo ==
pointer|| E1.tipo == null))){
            $.tipo = bool;
      }
      else $.tipo = error;
}
```

```
comprueba_tipos(desigual(E0, E1)){
      comprueba_tipos(E0);
      comprueba_tipos(E1);
      if (((E0.tipo == int || E0.tipo == real) && (E1.tipo == int ||
E1.tipo == real)) ||
      (E0.tipo == bool && E1.tipo == bool) ||
      (E0.tipo == string && E1.tipo == string) ||
      ((E0.tipo == pointer || E0.tipo == null) && (E1.tipo ==
pointer|| E1.tipo == null))){
            $.tipo = bool;
      }
      else $.tipo = error;
}

comprueba_tipos(mul(E0, E1)){
      comprueba_tipos(E0);
      comprueba_tipos(E1);
      if (E0.tipo == int && E1.tipo == int){
            $.tipo = int;
      }
      else if ((E0.tipo == real && (E1.tipo == real || E1.tipo ==
int)) || (E1.tipo == real && (E0.tipo == real || E0.tipo == int))){
            $.tipo = real;
      }
      else $.tipo = error;
}

comprueba_tipos(div(E0, E1)){
      comprueba_tipos(E0);
      comprueba_tipos(E1);
      if (E0.tipo == int && E1.tipo == int){
            $.tipo = int;
      }
      else if ((E0.tipo == real && (E1.tipo == real || E1.tipo ==
int)) || (E1.tipo == real && (E0.tipo == real || E0.tipo == int))){
            $.tipo = real;
      }
      else $.tipo = error;
}

comprueba_tipos(modulo(E0, E1)){
      comprueba_tipos(E0);
      comprueba_tipos(E1);
      if (E0.tipo == int && E1.tipo == int){
            $.tipo = int;
```

```
      }
      else $.tipo = error;
}

comprueba_tipos(m_unario(E)){
      comprueba_tipos(E);
      if (E.tipo == int) $.tipo = int;
      else if (E.tipo == real) $.tipo = real;
      else $.tipo = error;
}

comprueba_tipos(not(E)){
      comprueba_tipos(E);
      if (E.tipo == bool) $.tipo = bool;
      else $.tipo = error;
}

comprueba_tipos(indexacion(E0, E1)){
      comprueba_tipos(E0);
      comprueba_tipos(E1);
      if (E1.tipo == int && E0.tipo = tipo_array(num, T)){
            $.tipo = T;
      }
      else $.tipo = error;
}

comprueba_tipos(acc_registro(E, iden)){
      comprueba_tipos(E);
      if (E.tipo == registro) {
            $.tipo = E.campo(iden).tipo;
      } else {
            $.tipo = error;
      }
}

comprueba_tipos(acc_registro_ind(E, iden)){
      comprueba_tipos(E);
      if (E.tipo == pointer(registro)) {
            $.tipo = E.campo(iden).tipo;
      } else {
            $.tipo = error;
      }
}

comprueba_tipos(indireccion(E)){
      comprueba_tipos(E);
```

```
      if (E.tipo = tipo_pointer(T)){
            $.tipo = T;
      }
      else $.tipo = error;
}
-------------------------------------------------
compatibilidad(T0, T1){
      if ((T0 == int && T1 == int) ||
            (T0 == real && (T1 == int || T1 == real)) ||
            (T0 == bool && T1 == bool) ||
            (T0 == string && T1 == string) ||
            (T0 == pointer T && T1 == null) ||
            (T0 == pointer T && T1 == pointer T' && compatibilidad(T,
      T')) ||
            (T0 == array x of T && T1 == array x of T' &&
      compatibilidad(T, T')) ||
            ((T0 == record && T1 == record) && (T0.numeroCampos ==
      T1.numeroCampos) && foreach campo i; compatibilidad(T0.i,
      T1.i))){
            return true;
      }
      else return false;
}
```

PASO 3: Procesamiento de asignación de espacio

Nota: A partir de aquí no se han especificado ni implementado las funcionalidad de llamadas a procedimientos

```
global entero dir = 0
global entero nivel = 0

asigna_espacio(prog_sin_decs(Insts)) {
      asigna_espacio(Insts);
      $.tam = dir;
}
```

```
asigna_espacio(prog_con_decs(Decs, Insts)) {
      asigna_espacio(Decs);
      asigna_espacio(Insts);
      $.tam = dir; // Usado para crear la maquina p
}

asigna_espacio(dec_una(Dec)) {
      asigna_espacio(Dec);
}

asigna_espacio(decs_muchas(Decs, Dec)) {
      asigna_espacio(Decs, Dec);
}

asigna_espacio(var(Tipo, iden)){
      $.dir = dir
      $.nivel = nivel

      asigna_espacio(Tipo)

      $.tam = Tipo.tam
      $.tambase = Tipo.tambase
      dir += Tipo.tam
}

asigna_espacio(type(Tipo, iden)){
      $.tam = Tipo.tam
      $.tambase = Tipo.tambase
}

asigna_espacio(proc(iden, PFs, Bloque)){
      nivel++
      dir_ant = dir // Variable local
      dir = 0       // Direcciones locales al procedimiento (display)
      asigna_espacio(PFs)
      asigna_espacio(Bloque)
      $.nivel = nivel
      $.tam = dir
      dir = dir_ant
      nivel--
}

asigna_espacio(param_f_con_una(PF)){
      asigna_espacio(PF);
}
```

```
asigna_espacio(param_f_con_muchas(PFS, PF)){
      asigna_espacio(PFS);
      asigna_espacio(PF);
}

asigna_espacio(param_f_ref(Tipo, iden)){
      asigna_espacio(Tipo)
      $.dir = dir
      $.nivel = nivel
      dir += 1 // Tamaño de un puntero
}

asigna_espacio(param_f_noref(Tipo, iden)){
      asigna_espacio(Tipo)
      $.dir = dir
      $.nivel = nivel
      dir += Tipo.tam
}

asigna_espacio(Tipo_array(n, T)) {
      asigna_espacio(T);
      $.tambasesize = T.tam;
      $.tam = $.tambasesize * n;
}

asigna_espacio(Tipo_record(Cs)) {
      asigna_espacio(Cs);
      $.tam = Cs.tam;
}

asigna_espacio(Tipo_pointer(T)) {
      asigna_espacio(T);
      $.tam = 1;
}

asigna_espacio(Tipo_iden(T)) {
      tipo_iden.dir = T.vinculo.dir;
      tipo_iden.tam = T.vinculo.tam;
      tipo_iden.nivel = T.vinculo.nivel;
      tipo_iden.tambasesize = T.vinculo.tambasesize;
}

asigna_espacio(Tipo_int) {
      $.tam = 1
}
```

```
asigna_espacio(Tipo_real) {
      $.tam = 1
}

asigna_espacio(Tipo_bool) {
      $.tam = 1
}

asigna_espacio(Tipo_string) {
      $.tam = 1
}

asigna_espacio(Campos_Uno(C)) {
      asigna_espacio(C);
      $.tam = C.tam;
      C.despl = 0;
}

asigna_espacio(Campos_Muchos(Cs, C)) {
      asigna_espacio(Cs);
      asigna_espacio(C);
      C.despl = Cs.tam;
      $.tam = Cs.tam + C.tam;
}

asigna_espacio(Campo(T,id)) {
      asigna_espacio(T);
      $.tam = T.tam;
      $.basesize = $.basesize;
}

asigna_espacio(insts_muchas(Insts, Inst)){
      asigna_espacio(Insts);
      asigna_espacio(Inst);
}

asigna_espacio(inst_una(Inst)){
      asigna_espacio(Inst);
}

asigna_espacio(e_igual(E0, E1)){
      asigna_espacio(E0);
      asigna_espacio(E1);
}
```

```
asigna_espacio(if(E, PI)){
      asigna_espacio(E);
      asigna_espacio(PI);
}

asigna_espacio(ifelse(E, PI0, PI1)){
      asigna_espacio(E);
      asigna_espacio(PI0);
      asigna_espacio(PI1);
}

asigna_espacio(while(E, PI)){
      asigna_espacio(E);
      asigna_espacio(PI);
}

asigna_espacio(read(E)){
      asigna_espacio(E);
}


asigna_espacio(write(E)){
      asigna_espacio(E);
}

asigna_espacio(new(E)){
      asigna_espacio(E);
}

asigna_espacio(delete(E)){
      asigna_espacio(E);
}

asigna_espacio(lista_con(Insts)){
      asigna_espacio(Insts);
}

asigna_espacio(bloque_con(Prog)){
      asigna_espacio(Prog);
}

asigna_espacio(identificador(iden)){
      identificador.dir = identificador.vinculo.dir;
      identificador.tam = identificador.vinculo.tam;
      identificador.nivel = identificador.vinculo.nivel;
      identificador.tambase = identificador.vinculo.tambase;
```

```
}

asigna_espacio(suma(E0, E1)){
      asigna_espacio(E0);
      asigna_espacio(E1);
}

asigna_espacio(resta(E0, E1)){
      asigna_espacio(E0);
      asigna_espacio(E1);
}

asigna_espacio(and(E0, E1)){
      asigna_espacio(E0);
      asigna_espacio(E1);
}

asigna_espacio(or(E0, E1)){
      asigna_espacio(E0);
      asigna_espacio(E1);
}

asigna_espacio(menor(E0, E1)){
      asigna_espacio(E0);
      asigna_espacio(E1);
}

asigna_espacio(men_ig(E0, E1)){
      asigna_espacio(E0);
      asigna_espacio(E1);
}

asigna_espacio(mayor(E0, E1)){
      asigna_espacio(E0);
      asigna_espacio(E1);
}

asigna_espacio(may_ig(E0, E1)){
      asigna_espacio(E0);
      asigna_espacio(E1);
}

asigna_espacio(igual(E0, E1)){
      asigna_espacio(E0);
      asigna_espacio(E1);
}
```

```
asigna_espacio(desigual(E0, E1)){
      asigna_espacio(E0);
      asigna_espacio(E1);
}


asigna_espacio(mul(E0, E1)){
      asigna_espacio(E0);
      asigna_espacio(E1);
}

asigna_espacio(div(E0, E1)){
      asigna_espacio(E0);
      asigna_espacio(E1);
}

asigna_espacio(modulo(E0, E1)){
      asigna_espacio(E0);
      asigna_espacio(E1);
}

asigna_espacio(m_unario(E)){
      asigna_espacio(E)
}

asigna_espacio(not(E)){
      asigna_espacio(E);
}

asigna_espacio(indexacion(E0, E1)){
      asigna_espacio(E0);
      asigna_espacio(E1);
      // Usado por ejemplo en e_igual
      $.tam = E0.tambase;
}

asigna_espacio(acc_registro(E, iden)){
      asigna_espacio(E);
      $.tam = E.campo(iden).tam;
      $.tambase = E.campo(iden).tambase;
}

asigna_espacio(acc_registro_ind(E, iden)){
      asigna_espacio(E);
      $.tam = E.campo(iden).tam;
```

```
      $.tambase = E.campo(iden).tambase;
}

asigna_espacio(indireccion(E)){
      asigna_espacio(E);
      $.tam = 1;
      $.tambase = E.tam;
}
```

PASO 4: Repertorio de instrucciones de la máquina-p

Nota: Para mejorar la comprensión de las instrucciones, especificamos entre paréntesis los argumentos de las instrucciones. Además, entre corchetes especificamos los valores de la pila que usará la instrucción.

Instrucciones aritmético-lógicas	
`sumaint[b,a]`	Desapila b, desapila a, apila el resultado de a+b
`sumareal[b,a]`	Desapila b, desapila a, apila el resultado de a+b
`restaint[b,a]`	Desapila b, desapila a, apila el resultado de a-b
`restareal[b,a]`	Desapila b, desapila a, apila el resultado de a-b
`mulint[b,a]`	Desapila b, desapila a, apila el resultado de a*b
`mulreal[b,a]`	Desapila b, desapila a, apila el resultado de a*b
`divint[b,a]`	Desapila b, desapila a, apila el resultado de realizar la división entera de a entre b
`divreal[b,a]`	Despila b, desapila a, apila el resultado de realizar la división de a entre b
`modint[b,a]`	Desapila b, desapila a, apila el resto de la división entera de a entre b
`and[b,a]`	Desapila b, desapila a, apila true si ambos son true, o apila false en caso contrario
`or[b,a]`	Desapila b, desapila a, apila true si uno de ellos es true, o apila false en caso contrario.
`not[a]`	Desapila el booleano a, y apila su negación.
`intAReal[a]`	Desapila a, y apila una versión en formato real de a (hace un cast de entero a real)
`menorNum[b,a]`	Desapila b, desapila a, y apila true si a es estrictamente menor que b. En caso contrario apila false.

`menorIgualNum[b,a]`	Desapila b, desapila a, y apila true si a es menor o igual que b. Apila false en caso contrario.
`igualNum[b,a]`	Desapila b, desapila a, y apila true si a y b son iguales.
`menorString[b,a]`	Desapila b, desapila a, y apila true si a es estrictamente menor que b. En caso contrario apila false.
`menorIgualString[b,a]`	Desapila b, desapila a, y apila true si a es menor o igual que b. Apila false en caso contrario.
`igualString[b,a]`	Desapila b, desapila a, y apila true si a y b son iguales.
Instrucciones de movimiento de datos	
`apilaint(v)`	Apila el entero de valor *v* en la pila de evaluación
`apilareal(v)`	Apila el real de valor *v* en la pila de evaluación
`apilabool(v)`	Apila el booleano de valor *v* en la pila de evaluación
`apilastring(v)`	Apila la string de valor v en la pila de evaluación
`apilaind[d]`	Desapila una dirección *d* de la pila de evaluación, y apila en dicha pila el contenido de la celda *d* en la memoria de datos
`desapilaind[v,d]`	Desapila un valor *v* y una dirección *d* de la pila de evaluación (primero *v,* después *d*) y actualiza el contenido de la celda *d* en la memoria de datos a *v*.
`mueve(n)[`d_1`,`d_0`]`	Desapila dos direcciones d_1 y d_0 de la pila de evaluación (primero d_1 y luego d_0). Seguidamente copia el contenido de las *n* celdas consecutivas que comienzan en la dirección d_1 a las correspondientes *n* celdas que comienzan en la dirección d_0
Gestión de memoria dinámica	
`ira(d)`	Salta incondicionalmente a la dirección *d*
`irf(d)[v]`	Desapila un valor *v* de la pila de evaluación. Si es *falso* salta a *d*. Si no, continúa en secuencia.
`irv(d)[v]`	Desapila un valor v de la pila de evaluación. Si es *cierto* salta a *d*. Si no, continúa en secuencia.
`irind[d]`	Desapila una dirección *d* de la pila de evaluación y realiza un salto incondicional a dicha dirección.
Soporte a la ejecución de 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)[d]`	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 procedimineto 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[v]`	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)
`stop`	Detiene la máquina
Instrucciones de entrada/salida	
`writeint[d]`	Desapila la cima y la escribe en la salida estándar, como si fuese un entero.
`writereal[d]`	Desapila la cima y la escribe en la salida estándar, como si fuese un real.
`writebool[d]`	Desapila la cima y escribe true si el booleano tiene valor verdadero, false en caso contrario
`writestring[d]`	Desapila la cima y escribe el valor asociado en la salida estándar
`readint`	Apila en la cima un entero de la entrada estándar
`readreal`	Apila en la cima un real de la entrada estándar
`readstring`	Apila en la cima una línea de la entrada estándar

PASO 5: Procesamiento de etiquetado

```
global entero etq = 0

etiquetado(prog_sin_decs(Insts)) {
      $.etqi = etq;
      etiquetado(Insts);
      $.etqs = etq;
}

etiquetado(prog_con_decs(Decs, Insts)){
```

```
      $.etqi = etq;
      etiquetado(Insts);
      $.etqs = etq;
}

// Etiquetar las declaraciones solo tiene
// sentido si hubiesemos implementado procedimientos
etiquetado(decs_una(Dec)) {
      etiquetado(Dec);
}

etiquetado(decs_muchas(Decs,Dec)) {
      etiquetado(Decs);
      etiquedado(Dec);
}

etiquetado(insts_muchas(Insts, Inst)){
      $.etqi = etq;
      etiquetado(Insts);
      etiquetado(Inst);
      $.etqs = etq;
}

etiquetado(inst_una(Inst)){
      $.etqi = etq;
      etiquetado(Inst);
      $.etqs = etq;
}

etiquetado(e_igual(E0, E1)){
      $.etqi = etq;
      etiquetado(E1);
      etq++;
      $.etqs = etq;
}

etiquetado(if(E, PI)){
      $.etqi = etq;
      etiquetado(E);
      etq++;
      etiquetado(PI);
      $.etqs = etq;
}

etiquetado(ifelse(E, PI0, PI1)){
      $.etqi = etq;
```

```
      etiquetado(E);
      etq++;
      etiquetado(PI0);
      etq++;
      etiquetado(PI1);
      $.etqs = etq;
}

etiquetado(while(E, PI)){
      $.etqi = etq;
      etiquetado(E);
      etq++;
      etiquetado(PI);
      etq++;
      $.etqs = etq;
}

etiquetado(read(E)){
      $.etqi = etq;
      etiquetado(E);
      etq++; etq++;
      $.etqs = etq;

}


etiquetado(write(E)){
      $.etqi = etq;
      etiquetado(E);
      if (E.esDesignador()) etq++;
      etq++;
      $.etqs = etq;
}

etiquetado(nl()){
      $.etqi = etq;
      etq++; etq++;
      $.etqs = etq;
}

etiquetado(new(E)){
      $.etqi = etq;
      etiquetado(E);
      etq++; etq++;
      $.etqs = etq;
}
```

```
etiquetado(delete(E)){
      $.etqi = etq;
      etiquetado(E);
      etq++;
      $.etqs = etq;
}

etiquetado(call(iden, PR)){
      $.etqi = etq;
      etq++;
      etiquetado(PR);
      etq++;
      etq++;
      $.etqs = etq;

}

etiquetado(bl(Bloque)){
      $.etqi = etq;
      etiquetado(Bloque);
      $.etqs = etq;
}

etiquetado(lista_sin()){
      $.etqi = etq;
      $.etqs = etq;

}

etiquetado(lista_con(Insts)){
      $.etqi = etq;
      etiquetado(Insts);
      $.etqs = etq;
}

etiquetado(param_r_sin()){
      $.etqi = etq;
      $.etqs = etq;
}

etiquetado(param_r_con_una(E)){
      $.etqi = etq;
      etq++;
      etq++;
      etq++;
```

```
      etiquetado(E);
      etq++;
      $.etqs = etq;
}

etiquetado(param_r_con_muchas(PR, E)){
      $.etqi = etq;
      etiquetado(PR);
      etq++;
      etq++;
      etq++;
      etiquetado(E);
      etq++;
      $.etqs = etq;
}

etiquetado(bloque_sin()){
      $.etqi = etq;
      $.etqs = etq;
}

etiquetado(bloque_con(Prog)){
      $.etqi = etq;
      etiquetado(Prog);
      $.etqs = etq;
}

etiquetado(entero(num)){
      $.etqi = etq;
      etq++;
      $.etqs = etq;
}

etiquetado(real(num)){
      $.etqi = etq;
      etq++;
      $.etqs = etq;
}

etiquetado(cadena(iden)){
      $.etqi = etq;
      etq++;
      $.etqs = etq;
}

etiquetado(verdadero()){
```

```
	$.etqi = etq;
	etq++;
	$.etqs = etq;
}

etiquetado(falso()){
	$.etqi = etq;
	etq++;
	$.etqs = etq;
}

etiquetado(null()){
	$.etqi = etq;
	etq++;
	$.etqs = etq;
}

etiquetado(identificador(iden)){
	$.etqi = etq;
	etq++;
	if ($.vinculo == param_f_noref(T, iden) ||
!iden.esGlobal()){//esGlobal comprueba si iden es una variable global
o no
		etq++; etq++;
	}
	else if ($.vinculo == param_f_ref(T, iden)){
		etq++; etq++; etq++;
	}
	$.etqs = etq;
}

etiquetado(suma(E0, E1)){
	$.etqi = etq;
	etiquetado(E0);
	if (E0.esDesignador()) etq++;
	etiquetado(E1);
	if (E1.esDesignador()) etq++;
	etq++;
	$.etqs = etq;

}

etiquetado(resta(E0, E1)){
	$.etqi = etq;
	etiquetado(E0);
	if (E0.esDesignador()) etq++;
```

```
      etiquetado(E1);
      if (E1.esDesignador()) etq++;
      etq++;
      $.etqs = etq;
}

etiquetado(and(E0, E1)){
      $.etqi = etq;
      etiquetado(E0);
      if (E0.esDesignador()) etq++;
      etiquetado(E1);
      if (E1.esDesignador()) etq++;
      etq++;
      $.etqs = etq;
}

etiquetado(or(E0, E1)){
      $.etqi = etq;
      etiquetado(E0);
      if (E0.esDesignador()) etq++;
      etiquetado(E1);
      if (E1.esDesignador()) etq++;
      etq++;
      $.etqs = etq;
}

etiquetado(menor(E0, E1)){
      $.etqi = etq;
      etiquetado(E0);
      if (E0.esDesignador()) etq++;
      if (E0.tipo == Bool) etq++;
      etiquetado(E1);
      if (E1.esDesignador()) etq++;
      etq++;
      $.etqs = etq;
}

etiquetado(men_ig(E0, E1)){
      $.etqi = etq;
      etiquetado(E0);
      if (E0.esDesignador()) etq++;
      if (E0.tipo == Bool) etq++;
      etiquetado(E1);
      if (E1.esDesignador()) etq++;
      etq++;
      $.etqs = etq;
```

```
}

etiquetado(mayor(E0, E1)){
      $.etqi = etq;
      etiquetado(E0);
      if (E0.esDesignador()) etq++;
      if (E0.tipo == Bool) etq++;
      etiquetado(E1);
      if (E1.esDesignador()) etq++;
      etq++;
      $.etqs = etq;
}

etiquetado(may_ig(E0, E1)){
      $.etqi = etq;
      etiquetado(E0);
      if (E0.esDesignador()) etq++;
      if (E0.tipo == Bool) etq++;
      etiquetado(E1);
      if (E1.esDesignador()) etq++;
      etq++;
      $.etqs = etq;
}

etiquetado(igual(E0, E1)){
      $.etqi = etq;
      etiquetado(E0);
      if (E0.esDesignador()) etq++;
      etiquetado(E1);
      if (E1.esDesignador()) etq++;
      etq++;
      $.etqs = etq;
}

etiquetado(desigual(E0, E1)){
      $.etqi = etq;
      etiquetado(E0);
      if (E0.esDesignador()) etq++;
      etiquetado(E1);
      if (E1.esDesignador()) etq++;
      etq++; etq++;
      $.etqs = etq;
}

etiquetado(mul(E0, E1)){
      $.etqi = etq;
```

```
      etiquetado(E0);
      if (E0.esDesignador()) etq++;
      etiquetado(E1);
      if (E1.esDesignador()) etq++;
      etq++;
      $.etqs = etq;
}

etiquetado(div(E0, E1)){
      $.etqi = etq;
      etiquetado(E0);
      if (E0.esDesignador()) etq++;
      etiquetado(E1);
      if (E1.esDesignador()) etq++;
      etq++;
      $.etqs = etq;
}

etiquetado(modulo(E0, E1)){
      $.etqi = etq;
      etiquetado(E0);
      if (E0.esDesignador()) etq++;
      etiquetado(E1);
      if (E1.esDesignador()) etq++;
      etq++;
      $.etqs = etq;
}

etiquetado(m_unario(E)){
      $.etqi = etq;
      etq++;
      etiquetado(E);
      if (E.esDesignador()) etq++;
      etq++;
      $.etqs = etq;
}

etiquetado(not(E)){
      $.etqi = etq;
      etiquetado(E);
      if (E.esDesignador()) etq++;
      etq++;
      $.etqs = etq;
}

etiquetado(indexacion(E0, E1)){
```

```
      $.etqi = etq;
      etiquetado(E0);
      etiquetado(E1);
      if (E1.esDesignador()) etq++;
      etq++;
      etq++;
      etq++;
      $.etqs = etq;
}

etiquetado(acc_registro(E, iden)){
      $.etqi = etq;
      etiquetado(E);
      etq++;
      etq++;
      $.etqs = etq;
}

etiquetado(acc_registro_ind(E, iden)){
      $.etqi = etq;
      etiquetado(E);
      etq++;
      etq++;
      etq++;
      $.etqs = etq;
}

etiquetado(indireccion(E)){
      $.etqi = etq;
      etiquetado(E);
      etq++;
      $.etqs = etq;
}
```

PASO 6: Procesamiento de generación de código

```
genera_codigo(prog_sin_decs(Insts)) {
      genera_codigo(Insts);
}

genera_codigo(prog_con_decs(Decs, Insts)) {
      genera_codigo(Decs); // Codigo de procedimientos
      genera_codigo(Insts);
}

genera_codigo(dec_una(Dec)) {
      genera_codigo(Dec);
}

genera_codigo(decs_muchas(Decs, Dec)) {
      genera_codigo(Decs)
      genera_codigo(Dec);
}

genera_codigo(insts_muchas(Insts, Inst)){
      genera_codigo(Insts);
      genera_codigo(Inst);
}

genera_codigo(inst_una(Inst)){
      genera_codigo(Inst);
}



genera_codigo(e_igual(E0, E1)){
      genera_codigo(E0); // Tiene que ser un designador
      genera_codigo(E1);

      if (E1.esDesignador()) {
            genIns(mueve(E1.tam));
      } else {
            genIns(desapilaind);
      }
}

genera_codigo(if(E, PI)){
      genera_codigo(E);
      genIns(irf($.etqs)))
      genera_codigo(PI);
}
```

```
genera_codigo(ifelse(E, PI0, PI1)){
      genera_codigo(E);
      // Si da false, nos vamos al else
      genIns(irF(PI1.etqi));
      genera_codigo(PI0);
      // Después de la parte del if, nos saltamos la del else
      genIns(irA($.etqs));
      genera_codigo(PI1);
}

genera_codigo(while(E, PI)){
      genera_codigo(E);
      genera_codigo(PI);
}

genera_codigo(read(E)){
      genera_codigo(E);

      if (E.tipo == Tipo_Entero) {
            genIns(readInt);
      } else if (E.tipo == Tipo_Real) {
            genIns(readReal);
      } else if (E.tipo == Tipo_String) {
            genIns(readString);
      }

      genIns(desapilaInd);
}


genera_codigo(write(E)){
      genera_codigo(E);
      if (E.esDesignador()) genIns(apilaInd);

      if (E.tipo == Tipo_Entero) {
            genIns(writeInt);
      } else if (E.tipo == Tipo_Real) {
            genIns(writeReal);
      } else if (E.tipo == Tipo_Bool) {
            genIns(writeBool);
      } else if (E.tipo == Tipo_String) {
            genIns(writeString);
      }
}
```

```
genera_codigo(nl()){
      genInst(apilastring(“\n”))
      genInst(writestring)
}

genera_codigo(new(E)){
      genera_codigo(E);
      genInst(alloc(E.basesize));
      genInst(desapilaInd);
}

genera_codigo(delete(E)){
      genera_codigo(E);
      genInst(dealloc(E.basesize));
}

genera_codigo(bl(Bloque)){
      genera_codigo(Bloque);
}

genera_codigo(lista_con(Insts)){
      genera_codigo(Insts);
}

genera_codigo(bloque_con(Prog)){
      genera_codigo(Prog);
}

genera_codigo(entero(num)){
      genInst(apilaInt(num))
}

genera_codigo(real(num)){
      genInst(apilaReal(num))
}

genera_codigo(cadena(iden)){
      genInst(apilaString(iden))
}

genera_codigo(verdadero()){
      genInst(apilaBool(true))
}

genera_codigo(falso()){
      genInst(apilaBool(falso))
```

```
}

genera_codigo(null()){
      genInst(apilaInt(-1));
}

genera_codigo(identificador(iden)){
      if (identificador.nivel == 0) {
            // Variable global
            genIns(apilaInt(iden.dir));
      } else {
            genIns(apilad(iden.nivel));
            genIns(apilaInt(iden.dir));
            genIns(suma);
// Nota: Si es un parametro por referencia es necesario
// hacer un apilaind despues. Si lo usamos en una expresion tambien
      }
}

genera_codigo(suma(E0, E1)){
      genera_codigo(E0);
      if (E0.esDesignador()) genIns(apilaInd);
      genera_codigo(E1);
      if (E1.esDesignador()) genIns(apilaInd);
      if $.tipo == int:
            genIns(sumaint);
      else if $.tipo == real:
            genIns(sumareal);
}

genera_codigo(resta(E0, E1)){
      genera_codigo(E0);
      if (E0.esDesignador()) genIns(apilaInd);
      genera_codigo(E1);
      if (E1.esDesignador()) genIns(apilaInd);
      if $.tipo == int:
            genIns(restaint);
      else if $.tipo == real:
            genIns(restareal);
}

genera_codigo(and(E0, E1)){
      genera_codigo(E0);
      if (E0.esDesignador()) genIns(apilaInd);
      genera_codigo(E1);
      if (E1.esDesignador()) genIns(apilaInd);
```

```
      genIns(and);
}



genera_codigo(or(E0, E1)){
      genera_codigo(E0);
      if (E0.esDesignador()) genIns(apilaInd);
      genera_codigo(E1);
      if (E1.esDesignador()) genIns(apilaInd);
      genIns(or);
}

genera_codigo(menor(E0, E1)){
      // Nota: En el caso de bool
      // a < b <-> !a ^ b
      genera_codigo(E0);
      if (E0.esDesignador()) genIns(apilaInd);
      if (E0.tipo == bool) { genIns(not); }
      genera_codigo(E1);
      if (E1.esDesignador()) genIns(apilaInd);

      if (E0.tipo.isNum()) {
            genIns(menorNum);
      } else if (E0.tipo == String) {
            genIns(menorString);
      } else if (E0.tipo == Bool) {
            genIns(and);
      }
}

genera_codigo(men_ig(E0, E1)){
      // Nota: En el caso de bool
      // a < b <-> !a v b
      genera_codigo(E0);
      if (E0.esDesignador()) genIns(apilaInd);
      if (E0.tipo == bool) { genIns(not); }
      genera_codigo(E1);
      if (E1.esDesignador()) genIns(apilaInd);

      if (E0.tipo.isNum()) {
            genIns(menorIgNum);
      } else if (E0.tipo == String) {
            genIns(menorIgString);
      } else if (E0.tipo == Bool) {
            genIns(or);
```

```
      }
}

genera_codigo(mayor(E0, E1)){
      // Nota: a > b <-> b < a

      genera_codigo(E1);
      if (E1.esDesignador()) genIns(apilaInd);
      if (E0.tipo == bool) { genIns(not); }
      genera_codigo(E0);
      if (E0.esDesignador()) genIns(apilaInd);

      if (E0.tipo.isNum()) {
            genIns(menorNum);
      } else if (E0.tipo == String) {
            genIns(menorString);
      } else if (E0.tipo == Bool) {
            genIns(and);
      }
}

genera_codigo(may_ig(E0, E1)){
      genera_codigo(E1);
      if (E1.esDesignador()) genIns(apilaInd);
      if (E0.tipo == bool) { genIns(not); }
      genera_codigo(E0);
      if (E0.esDesignador()) genIns(apilaInd);

      if (E0.tipo.isNum()) {
            genIns(menorIgNum);
      } else if (E0.tipo == String) {
            genIns(menorIgString);
      } else if (E0.tipo == Bool) {
            genIns(or);
      }
}

genera_codigo(igual(E0, E1)){
      genera_codigo(E0);
      if (E0.esDesignador()) genIns(apilaInd);
      genera_codigo(E1);
      if (E1.esDesignador()) genIns(apilaInd);

      if (E0.tipo.isNum()) {
            genIns(igualNum);
      } else if (E0.tipo == String) {
```

```
            genIns(igualString);
      } else if (E0.tipo == Bool) {
            // Con bool a == b <-> a ^ b
            genIns(and);
      }
}

genera_codigo(desigual(E0, E1)){
      genera_codigo(E0);
      if (E0.esDesignador()) genIns(apilaInd);
      genera_codigo(E1);
      if (E1.esDesignador()) genIns(apilaInd);

      if (E0.tipo.isNum()) {
            genIns(igualNum);
      } else if (E0.tipo == String) {
            genIns(igualString);
      } else if (E0.tipo == Bool) {
            // Con bool a == b <-> a ^ b
            genIns(and);
      }

      genIns(not);
}


genera_codigo(mul(E0, E1)){
      genera_codigo(E0);
      if (E0.esDesignador()) genIns(apilaInd);
      genera_codigo(E1);
      if (E1.esDesignador()) genIns(apilaInd);
      if $.tipo == int:
            genIns(mulint);
      else if $.tipo == real:
            genIns(mulreal);
}

genera_codigo(div(E0, E1)){
      genera_codigo(E0);
      if (E0.esDesignador()) genIns(apilaInd);
      genera_codigo(E1);
      if (E1.esDesignador()) genIns(apilaInd);
      if $.tipo == int:
            genIns(divint);
      else if $.tipo == real:
            genIns(divreal);
```

```
}

genera_codigo(modulo(E0, E1)){
      genera_codigo(E0);
      if (E0.esDesignador()) genIns(apilaInd);
      genera_codigo(E1);
      if (E1.esDesignador()) genIns(apilaInd);
      genIns(modulo);
}

genera_codigo(m_unario(E)){
      if $.tipo == int:
            genIns(apilaint(0));
            genera_codigo(E);
            if (E.esDesignador()) genIns(apilaInd);
            genIns(restaint);
      else if $.tipo == real:
            genIns(apilareal(0.0));
            genera_codigo(E);
            if (E.esDesignador()) genIns(apilaInd);
            genIns(restareal);
}

genera_codigo(not(E)){
      genera_codigo(E);
      if (E.esDesignador()) genIns(apilaInd);
      genIns(not);
}

genera_codigo(indexacion(E0, E1)){
      genera_codigo(E0);
      genera_codigo(E1);
      if (E1.esDesignador()) genIns(apilaInd);
      genIns(apilaInt(E0.tambase));
      genIns(mul);
      genIns(suma);
}

genera_codigo(acc_registro(E, iden)){
      genera_codigo(E);
      genIns(apilaInt(E.campo(iden).despl));
      genIns(suma);
}

genera_codigo(acc_registro_ind(E, iden)){
      genera_codigo(E);
```

```
      genIns(apilaInd);
      genIns(apilaInt(E.campo(iden.despl));
      genIns(suma);
}

genera_codigo(indireccion(E)){
      genera_codigo(E);
      genIns(apilaInd);
}
```