
Procesadores de Lenguajes

Memoria de proyecto — Hito 4: Finalización del procesador

GRUPO 14

RODRIGO SOUTO SANTOS
LEONARDO PRADO DE SOUZA
JUAN ANDRÉS HIBJAN CARDONA
IZAN RODRIGO SANZ

*Grado en Ingeniería informática
Facultad de Informática
Universidad Complutense de Madrid*



Índice general

1. Especificación del Procesamiento de Vinculación	2
1.1. Funciones para la tabla de símbolos	2
1.2. Funciones de procesamiento	2
1.2.1. Declaraciones	2
1.2.2. Tipos	4
1.2.3. Instrucciones	5
1.2.4. Expresiones	6
2. Especificación del Procesamiento de Pre-tipado	9
2.1. Funciones para el conjunto de campos	9
2.2. Funciones de procesamiento	9
2.2.1. Declaraciones	9
2.2.2. Tipos	10
2.2.3. Instrucciones	11
3. Especificación del Procesamiento de Comprobación de Tipos	12
3.1. Funciones para el conjunto de pares de tipos	12
3.2. Funciones de procesamiento	12
3.2.1. Declaraciones	12
3.2.2. Instrucciones	12
3.2.3. Expresiones	15
3.3. Funciones auxiliares	19
4. Especificación del Procesamiento de Asignación de Espacio	22
4.1. Funciones de procesamiento	22
4.1.1. Declaraciones	22
4.1.2. Tipos	24
4.1.3. Instrucciones	25
4.2. Funciones auxiliares	26
5. Descripción de las Instrucciones de la Máquina-p	27
5.1. Operaciones	27
5.2. Tipos básicos	27
5.3. Direccionamiento	27
5.4. Procedimientos	28
5.5. Anidamiento	28
5.6. Saltos	28
5.7. I/O	28
5.8. Otros	28
6. Especificación del Procesamiento de Etiquetado	29
6.1. Funciones para la pila de procedimientos	29
6.2. Funciones de procesamiento	29
6.2.1. Declaraciones	29
6.2.2. Instrucciones	30
6.2.3. Expresiones	31
6.3. Funciones auxiliares	34
7. Especificación del Procesamiento de Generación de Código	36
7.1. Funciones para la pila de procedimientos	36
7.2. Funciones de procesamiento	36
7.2.1. Declaraciones	36
7.2.2. Instrucciones	37
7.2.3. Expresiones	38
7.3. Funciones auxiliares	40

1 | Especificación del Procesamiento de Vinculación

1.1. Funciones para la tabla de símbolos

- **creaTS()**: Crea una tabla de símbolos que no tiene aún ningún ámbito abierto.
- **abreAmbito(ts)**: Añade a la tabla de símbolos **ts** un nuevo ámbito, que tendrá como padre el ámbito más reciente (o \perp , si aún no se ha creado ningún ámbito).
- **contiene(ts,id)**: Comprueba si el ámbito actual de la tabla de símbolos **ts** contiene ya una entrada para el identificador **id**.
- **inserta(ts,id,dec)**: Inserta el identificador **id** en el ámbito actual de la tabla de símbolos **ts**, con la referencia al nodo **dec** como valor.
- **vinculoDe(ts,id)**: Recupera la referencia asociada a **id** en la tabla de símbolos **ts**. Para ello busca sucesivamente en la cadena de ámbitos, hasta que lo encuentra (si no está, devuelve \perp).
- **cierraAmbito(ts)**: Fija en **ts** el ámbito actual al ámbito padre del ámbito más reciente.

1.2. Funciones de procesamiento

```
var ts = crearTS()
vincula(bloque(SecDecs, SecIs)) :
  abreAmbito(ts)
  vincula(SecDecs)
  vincula(SecIs)
  cierraAmbito(ts)
```

1.2.1. Declaraciones

```
vincula(si_decs(LDecs)) :
  vincula1(LDecs)
  vincula2(LDecs)

vincula(no_decs()) : noop

vincula1(muchas_decs(LDecs, Dec)) :
  vincula1(LDecs)
  vincula1(Dec)

vincula2(muchas_decs(LDecs, Dec)) :
  vincula2(LDecs)
  vincula2(Dec)

vincula1(una_dec(Dec)) :
  vincula1(Dec)

vincula2(una_dec(Dec)) :
  vincula2(Dec)

vincula1(dec_base(TipoNom)) :
  let TipoNom = TipoNom(Tipo, iden) in
```

```

        if contiene(ts, iden) then
            error
        else
            inserta(ts, iden, $)
        endif
    end let
vincula1(TipoNom)

vincula2(dec_base(TipoNom)) :
    vincula2(TipoNom)

vincula1(dec_type(TipoNom)) :
    let TipoNom = TipoNom(Tipo, iden) in
        if contiene(ts, iden) then
            error
        else
            inserta(ts, iden, $)
        endif
    end let
vincula1(TipoNom)

vincula2(dec_type(TipoNom)) :
    vincula2(TipoNom)

vincula1(dec_proc(iden, ParamFs, Bloq)) :
    if contiene(ts, iden) then
        error
    else
        inserta(ts, iden, $)
    endif

vincula2(dec_proc(iden, ParamFs, Bloq)) :
    abreAmbito(ts)
    vincula1(ParamFs)
    vincula2(ParamFs)
    vincula(Bloq)
    cierraAmbito(ts)

vincula1(si_params_f(LParamFs)) :
    vincula1(LParamFs)

vincula2(si_params_f(LParamFs)) :
    vincula2(LParamFs)

vincula1(no_params_f()) : noop

vincula2(no_params_f()) : noop

vincula1(muchos_params_f(LParamFs, ParamF)) :
    vincula1(LParamFs)
    vincula1(ParamF)

vincula2(muchos_params_f(LParamFs, ParamF)) :
    vincula2(LParamFs)
    vincula2(ParamF)

vincula1(un_param_f(ParamF)) :
    vincula1(ParamF)

vincula2(un_param_f(ParamF)) :

```

vincula2(*ParamF*)

```
vincula1(si_refparam_f(Tipo, iden)) :
  if contiene(ts, iden) then
    error
  else
    inserta(ts, iden, $)
  endif
vincula1(Tipo)
```

```
vincula2(si_refparam_f(Tipo, iden)) :
  vincula2(Tipo)
```

```
vincula1(no_refparam_f(Tipo, iden)) :
  if contiene(ts, iden) then
    error
  else
    inserta(ts, iden, $)
  endif
vincula1(Tipo)
```

```
vincula2(no_refparam_f(Tipo, iden)) :
  vincula2(Tipo)
```

1.2.2. Tipos

```
vincula1(tipo_nombre(Tipo, iden)) :
  vincula1(Tipo)
```

```
vincula2(tipo_nombre(Tipo, iden)) :
  vincula2(Tipo)
```

```
vincula1(tipo_array(Tipo, litEntero)) :
  if Tipo != tipo_type(_) then
    vincula1(Tipo)
  endif
```

```
vincula2(tipo_array(Tipo, litEntero)) :
  if Tipo == tipo_type(iden) then
    Tipo.vinculo = vinculoDe(ts, iden)
    if Tipo.vinculo != dec_type(_) then
      error
    endif
  else
    vincula2(Tipo)
  endif
```

```
vincula1(tipo_indir(Tipo)) :
  if Tipo != tipo_type(_) then
    vincula1(Tipo)
  endif
```

```
vincula2(tipo_indir(Tipo)) :
  if Tipo == tipo_type(iden) then
    Tipo.vinculo = vinculoDe(ts, iden)
    if Tipo.vinculo != dec_type(_) then
      error
    endif
  else
    vincula2(Tipo)
```

endif

vincula1(tipo_struct(LCampos)) :
vincula1(LCampos)

vincula2(tipo_struct(LCampos)) :
vincula2(LCampos)

vincula1(tipo_int()) : noop

vincula2(tipo_int()) : noop

vincula1(tipo_real()) : noop

vincula2(tipo_real()) : noop

vincula1(tipo_bool()) : noop

vincula2(tipo_bool()) : noop

vincula1(tipo_string()) : noop

vincula2(tipo_string()) : noop

vincula1(tipo_type(iden)) : noop

vincula2(tipo_type(iden)) :
\$.vinculo = vinculoDe(ts, iden)

vincula1(muchos_campos(LCampos, TipoNom)) :
vincula1(LCampos)
vincula1(TipoNom)

vincula2(muchos_campos(LCampos, TipoNom)) :
vincula2(LCampos)
vincula2(TipoNom)

vincula1(un_campo(TipoNom)) :
vincula1(TipoNom)

vincula2(un_campo(TipoNom)) :
vincula2(TipoNom)

1.2.3. Instrucciones

vincula(si_ins(LIs)) :
vincula(LIs)

vincula(no_ins()) : noop

vincula(muchas_ins(LIs, I)) :
vincula(LIs)
vincula(I)

vincula(una_ins(I)) :
vincula(I)

vincula(ins_eval(Exp)) :
vincula(Exp)

```

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

vincula(ins_if_else(I, Bloq)) :
    vincula(I)
    vincula(Bloq)

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

vincula(ins_read(Exp)) :
    vincula(Exp)

vincula(ins_write(Exp)) :
    vincula(Exp)

vincula(ins_nl()) : noop

vincula(ins_new(Exp)) :
    vincula(Exp)

vincula(ins_delete(Exp)) :
    vincula(Exp)

vincula(ins_call(iden, ParamRs)) :
    $.vinculo = vinculoDe(ts, iden)
    if $.vinculo ==  $\perp$  then
        error
    endif
    vincula(ParamRs)

vincula(ins_bloque(Bloq)) :
    vincula(Bloq)

vincula(si_params_r(LParamRs)) :
    vincula(LParamRs)

vincula(no_params_r()) : noop

vincula(muchos_params_r(LParamRs, Exp)) :
    vincula(LParamRs)
    vincula(Exp)

vincula(un_param_r(Exp)) :
    vincula(Exp)

```

1.2.4. Expresiones

```

vincula(exp_asig(Opnd0, Opnd1)) :
    vinculaExpBin(Opnd0, Opnd1)

vincula(exp_menor(Opnd0, Opnd1)) :
    vinculaExpBin(Opnd0, Opnd1)

vincula(exp_menor_ig(Opnd0, Opnd1)) :
    vinculaExpBin(Opnd0, Opnd1)

vincula(exp_mayor(Opnd0, Opnd1)) :

```

```

    vinculaExpBin(Opnd0, Opnd1)

vincula(exp_mayor_ig(Opnd0, Opnd1)) :
    vinculaExpBin(Opnd0, Opnd1)

vincula(exp_ig(Opnd0, Opnd1)) :
    vinculaExpBin(Opnd0, Opnd1)

vincula(exp_dist(Opnd0, Opnd1)) :
    vinculaExpBin(Opnd0, Opnd1)

vincula(exp_suma(Opnd0, Opnd1)) :
    vinculaExpBin(Opnd0, Opnd1)

vincula(exp_resta(Opnd0, Opnd1)) :
    vinculaExpBin(Opnd0, Opnd1)

vincula(exp_and(Opnd0, Opnd1)) :
    vinculaExpBin(Opnd0, Opnd1)

vincula(exp_or(Opnd0, Opnd1)) :
    vinculaExpBin(Opnd0, Opnd1)

vincula(exp_mul(Opnd0, Opnd1)) :
    vinculaExpBin(Opnd0, Opnd1)

vincula(exp_div(Opnd0, Opnd1)) :
    vinculaExpBin(Opnd0, Opnd1)

vincula(exp_mod(Opnd0, Opnd1)) :
    vinculaExpBin(Opnd0, Opnd1)

vincula(exp_menos(Opnd)) :
    vincula(Opnd)

vincula(exp_not(Opnd)) :
    vincula(Opnd)

vincula(exp_index(Opnd0, Opnd1)) :
    vincula(Opnd0)
    vincula(Opnd1)

vincula(exp_reg(Opnd, iden)) :
    vincula(Opnd)

vincula(exp_indir(Opnd)) :
    vincula(Opnd)

vincula(exp_entero(litEntero)) : noop

vincula(exp_real(litReal)) : noop

vincula(exp_true()) : noop

vincula(exp_false()) : noop

vincula(exp_cadena(litCadena)) : noop

vincula(exp_iden(iden)) :
    $.vinculo = vinculoDe(ts, iden)

```



```
if $.vinculo ==  $\perp$  then  
    error  
endif
```

```
vincula(exp_null()) : noop
```

```
vinculaExpBin(Opnd0, Opnd1) :  
    vincula(Opnd0)  
    vincula(Opnd1)
```

2 | Especificación del Procesamiento de Pre-tipado

2.1. Funciones para el conjunto de campos

- `nuevoConjunto()`: Crea un conjunto vacío para almacenar los campos de un registro
- `contiene(set,id)`: Comprueba si el conjunto `set` contiene ya una entrada para el identificador `id`.
- `inserta(set,id)`: Inserta el identificador `id` en el conjunto `set`.

2.2. Funciones de procesamiento

`var set`

```
pretipado(bloque(SecDecs, SecIs)) :
    pretipado(SecDecs)
    pretipado(SecIs)
```

2.2.1. Declaraciones

```
pretipado(si_decs(LDecs)) :
    pretipado(LDecs)
```

```
pretipado(no_decs()) : noop
```

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

```
pretipado(una_dec(Dec)) :
    pretipado(Dec)
```

```
pretipado(dec_base(TipoNom)) :
    pretipado(TipoNom)
```

```
pretipado(dec_type(TipoNom)) :
    pretipado(TipoNom)
```

```
pretipado(dec_proc(iden, ParamFs, Bloq)) :
    pretipado(ParamFs)
    pretipado(Bloq)
```

```
pretipado(si_params_f(LParamFs)) :
    pretipado(LParamFs)
```

```
pretipado(no_params_f()) : noop
```

```
pretipado(muchos_params_f(LParamFs, ParamF)) :
    pretipado(LParamFs)
    pretipado(ParamF)
```

```
pretipado(un_param_f(ParamF)) :
    pretipado(ParamF)
```

```
pretipado(si_refparam_f(Tipo, iden)) :
    pretipado(Tipo)
```

```
pretipado(no_refparam_f(Tipo, iden)) :
    pretipado(Tipo)
```

2.2.2. Tipos

```
pretipado(tipo_nombre(Tipo, iden)) :
    pretipado(Tipo)
```

```
pretipado(tipo_array(Tipo, litEntero)) :
    if litEntero < 0 then
        error
    endif
    pretipado(Tipo)
```

```
pretipado(tipo_indir(Tipo)) :
    pretipado(Tipo)
```

```
pretipado(tipo_struct(LCampos)) :
    var tmp = set
    set = nuevoConjunto()
    pretipado(LCampos)
    set = tmp
```

```
pretipado(tipo_int()) : noop
```

```
pretipado(tipo_real()) : noop
```

```
pretipado(tipo_bool()) : noop
```

```
pretipado(tipo_string()) : noop
```

```
pretipado(tipo_type(iden)) :
    if $.vinculo != dec_type(_) then
        error
    endif
```

```
pretipado(muchos_campos(LCampos, TipoNom(Tipo, iden))) :
    pretipado(LCampos)
    pretipado(TipoNom)
    if contiene(set, iden) then
        error
    else
        inserta(set, iden)
    endif
```

```
pretipado(un_campo(TipoNom(Tipo, iden))) :
    pretipado(TipoNom)
    if contiene(set, iden) then
        error
    else
        inserta(set, iden)
    endif
```

2.2.3. Instrucciones

pretipado(**si_ins**(*LIs*)) :
 pretipado(*LIs*)

pretipado(**no_ins**()) : **noop**

pretipado(**muchas_ins**(*LIs*, *I*)) :
 pretipado(*LIs*)
 pretipado(*I*)

pretipado(**una_ins**(*I*)) :
 pretipado(*I*)

pretipado(**ins_eval**(*Exp*)) : **noop**

pretipado(**ins_if**(*Exp*, *Bloq*)) :
 pretipado(*Bloq*)

pretipado(**ins_if_else**(*I*, *Bloq*)) :
 pretipado(*I*)
 pretipado(*Bloq*)

pretipado(**ins_while**(*Exp*, *Bloq*)) :
 pretipado(*Bloq*)

pretipado(**ins_read**(*Exp*)) : **noop**

pretipado(**ins_write**(*Exp*)) : **noop**

pretipado(**ins_nl**()) : **noop**

pretipado(**ins_new**(*Exp*)) : **noop**

pretipado(**ins_delete**(*Exp*)) : **noop**

pretipado(**ins_call**(*iden*, *ParamRs*)) : **noop**

pretipado(**ins_bloque**(*Bloq*)) :
 pretipado(*Bloq*)

3 | Especificación del Procesamiento de Comprobación de Tipos

3.1. Funciones para el conjunto de pares de tipos

- `nuevoConjunto()`: Crea un conjunto vacío para almacenar los pares de tipos
- `contiene(set, tipo0, tipo1)`: Comprueba si el conjunto `set` contiene ya una entrada para el par (`tipo0`, `tipo1`).
- `inserta(set, tipo0, tipo1)`: Inserta el par (`tipo0`, `tipo1`) en el conjunto `set`.

3.2. Funciones de procesamiento

var set

```
tipado(bloque(SecDecs, SecIs)) :
  tipado(SecDecs)
  tipado(SecIs)
  $.tipo = ambos_ok(SecDecs.tipo, SecIs.tipo)
```

3.2.1. Declaraciones

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

```
tipado(no_decs()) :
  $.tipo = ok
```

```
tipado(muchas_decs(LDecs, Dec)) :
  tipado(LDecs)
  tipado(Dec)
  $.tipo = ambos_ok(LDecs.tipo, Dec.tipo)
```

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

```
tipado(dec_base(TipoNom)) :
  $.tipo = ok
```

```
tipado(dec_type(TipoNom)) :
  $.tipo = ok
```

```
tipado(dec_proc(iden, ParamFs, Bloq)) :
  tipado(Bloq)
  $.tipo = Bloq.tipo
```

3.2.2. Instrucciones

```
tipado(si_ins(LIs)) :
  tipado(LIs)
```

```

    $.tipo = LI.tipo

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

tipado(muchas_ins(LIs, I)) :
    tipado(LIs)
    tipado(I)
    $.tipo = ambos_ok(LI.tipo, I.tipo)

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

tipado(ins_eval(Exp)) :
    tipado(Exp)
    if Exp.tipo != error then
        $.tipo = ok
    else
        $.tipo = error
    endif

tipado(ins_if(Exp, Bloq)) :
    tipado(Exp)
    tipado(Bloq)
    if ref!(Exp.tipo) == tipo_bool() ∧ Bloq.tipo == ok then
        $.tipo = ok
    else
        $.tipo = error
    error
    endif

tipado(ins_if_else(I, Bloq)) :
    tipado(I)
    tipado(Bloq)
    $.tipo = ambos_ok(I.tipo, Bloq.tipo)

tipado(ins_while(Exp, Bloq)) :
    tipado(Exp)
    tipado(Bloq)
    if ref!(Exp.tipo) == tipo_bool() ∧ Bloq.tipo == ok then
        $.tipo = ok
    else
        $.tipo = error
    error
    endif

tipado(ins_read(Exp)) :
    tipado(Exp)
    let T = ref!(Exp.tipo) in
        if T == tipo_int() ∨
           T == tipo_real() ∨
           T == tipo_string() then
            if es_designador(Exp) then
                $.tipo = ok
            else
                $.tipo = error
            error
        endif
    else

```

```

        $.tipo = error
    error
endif
end let

tipado(ins_write(Exp)) :
    tipado(Exp)
    let T = ref!(Exp.tipo) in
        if (T == tipo_int() ∨
            T == tipo_real() ∨
            T == tipo_bool() ∨
            T == tipo_string()) then
            $.tipo = ok
        else
            $.tipo = error
        error
    endif
end let

tipado(ins_nl()) :
    $.tipo = ok

tipado(ins_new(Exp)) :
    tipado(Exp)
    if ref!(Exp.tipo) == tipo_indir() then
        $.tipo = ok
    else
        $.tipo = error
    error
endif

tipado(ins_delete(Exp)) :
    tipado(Exp)
    if ref!(Exp.tipo) == tipo_indir() then
        $.tipo = ok
    else
        $.tipo = error
    error
endif

tipado(ins_call(iden, ParamRs)) :
    tipado(ParamRs)
    if $.vinculo != dec_proc(_, ParamFs, _) then
        if ParamRs == no_params_r() ∧ ParamFs == no_params_f() then
            $.tipo = ok
        else if ParamRs == si_params_r(LParamRs) ∧ ParamFs == si_params_f(LParamFs) then
            if num_params(LParamRs, LParamFs) then
                if compatibles_params(LParamRs, LParamFs) then
                    $.tipo = ok
                else
                    $.tipo = error
                endif
            else
                $.tipo = error
            error
        endif
    else
        $.tipo = error
    error
endif
end let

```

```

    else
        $.tipo = error
    error
endif

tipado(ins_bloque(Bloq)) :
    tipado(Bloq)
    $.tipo = Bloq.tipo

tipado(si_params_r(LParamRs)) :
    tipado(LParamRs)

tipado(no_params_r()) : noop

tipado(muchos_params_r(LParamRs, Exp)) :
    tipado(LParamRs)
    tipado(Exp)

tipado(un_param_r(Exp)) :
    tipado(Exp)

```

3.2.3. Expresiones

```

tipado(exp_asig(Opnd0, Opnd1)) :
    tipado(Opnd0)
    tipado(Opnd1)
    if es_designador(Opnd0) then
        if compatibles(Opnd0.tipo, Opnd1.tipo) then
            $.tipo = ok
        else
            $.tipo = error
        error
    endif
else
    $.tipo = error
error
endif

tipado(exp_menor(Opnd0, Opnd1)) :
    tipado_rel(Opnd0, Opnd1, $)

tipado(exp_menor_ig(Opnd0, Opnd1)) :
    tipado_rel(Opnd0, Opnd1, $)

tipado(exp_mayor(Opnd0, Opnd1)) :
    tipado_rel(Opnd0, Opnd1, $)

tipado(exp_mayor_ig(Opnd0, Opnd1)) :
    tipado_rel(Opnd0, Opnd1, $)

tipado(exp_ig(Opnd0, Opnd1)) :
    tipado_rel(Opnd0, Opnd1, $)

tipado(exp_dist(Opnd0, Opnd1)) :
    tipado_rel(Opnd0, Opnd1, $)

tipado(exp_suma(Opnd0, Opnd1)) :
    tipado_arit(Opnd0, Opnd1, $)

tipado(exp_resta(Opnd0, Opnd1)) :

```



```

    tipado_arit(Opnd0, Opnd1, $)

tipado(exp_and(Opnd0, Opnd1)) :
    tipado_logic(Opnd0, Opnd1, $)

tipado(exp_or(Opnd0, Opnd1)) :
    tipado_logic(Opnd0, Opnd1, $)

tipado(exp_mul(Opnd0, Opnd1)) :
    tipado_arit(Opnd0, Opnd1, $)

tipado(exp_div(Opnd0, Opnd1)) :
    tipado_arit(Opnd0, Opnd1, $)

tipado(exp_mod(Opnd0, Opnd1)) :
    tipado(Opnd0)
    tipado(Opnd1)
    if ref!(Opnd0.tipo) == tipo_int() ∧
        ref!(Opnd1.tipo) == tipo_int() then
        $.tipo = tipo_int()
    else
        $.tipo = error
        aviso_error_bin(T0, T1)
    endif

tipado(exp_menos(Opnd)) :
    tipado(Opnd)
    let T = ref!(Opnd.tipo) in
        if T == tipo_int() then
            $.tipo = tipo_int()
        else if T == tipo_real() then
            $.tipo = tipo_real()
        else
            $.tipo = error
            aviso_error_un(T)
        endif
    end let

tipado(exp_not(Opnd)) :
    tipado(Opnd)
    if ref!(Opnd.tipo) == tipo_bool() then
        $.tipo = tipo_bool()
    else
        $.tipo = error
        aviso_error_un(T)
    endif

tipado(exp_index(Opnd0, Opnd1)) :
    tipado(Opnd0)
    tipado(Opnd1)
    if ref!(Opnd0.tipo) == tipo_array(Tipo, _) ∧
        ref!(Opnd1.tipo) == tipo_int() then
        $.tipo = tipo_array(Tipo, _)
    else
        $.tipo = error
        error
    endif

tipado(exp_reg(Opnd, iden)) :
    tipado(Opnd)

```

```

    if ref!(Opnd.tipo) == tipo_struct(LCampos) then
        let C = campo_struct(LCampos, iden) in
            if C == error then
                error
            endif
            $.tipo = C
        end let
    else
        $.tipo = error
        error
    endif

tipado(exp_indir(Opnd)) :
    tipado(Opnd)
    if ref!(Opnd.tipo) == tipo_indir(Tipo) then
        $.tipo = tipo_indir(Tipo)
    else
        $.tipo = error
        error
    endif

tipado(exp_entero(litEntero)) :
    $.tipo = tipo_int()

tipado(exp_real(litReal)) :
    $.tipo = tipo_real()

tipado(exp_true()) :
    $.tipo = tipo_bool()

tipado(exp_false()) :
    $.tipo = tipo_bool()

tipado(exp_cadena(litCadena)) :
    $.tipo = tipo_string()

tipado(exp_iden(iden)) :
    if $.vinculo = dec_base(TipoNom) then
        let TipoNom = TipoNom(Tipo, iden) in
            $.tipo = Tipo
        end let
    else if $.vinculo = si_refparam_f(Tipo, _) ∨ $.vinculo = no_refparam_f(Tipo, _) then
        $.tipo = Tipo
    else
        $.tipo = error
        error
    endif

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

tipado_arit(Opnd0, Opnd1, Exp) :
    tipado(Opnd0)
    tipado(Opnd1)
    let T0 = ref!(Opnd0.tipo) ∧ T1 = ref!(Opnd1.tipo) in
        if T0 == tipo_int() ∧
            T1 == tipo_int() then
            Exp.tipo = tipo_int()
        else if (T0 == tipo_real() ∨
            T0 == tipo_int()) ∧

```

```

        (T1 == tipo_real() ∨
         T1 == tipo_int()) then
            Exp.tipo = tipo_real()
        else
            Exp.tipo = error
            aviso_error_bin(T0, T1)
        endif
    end let

tipado_logic(Opnd0, Opnd1, Exp) :
    tipado(Opnd0)
    tipado(Opnd1)
    let T0 = ref!(Opnd0.tipo) ∧ T1 = ref!(Opnd1.tipo) in
        if T0 == tipo_bool() ∧
           T1 == tipo_bool() then
            Exp.tipo = tipo_bool()
        else
            Exp.tipo = error
            aviso_error_bin(T0, T1)
        endif
    end let

tipado_rel(Opnd0, Opnd1, Exp) :
    tipado(Opnd0)
    tipado(Opnd1)
    let T0 = ref!(Opnd0.tipo) ∧ T1 = ref!(Opnd1.tipo) in
        if (T0 == tipo_int() ∨
            T0 == tipo_real()) ∧
           (T1 == tipo_int() ∨
            T1 == tipo_real()) then
            Exp.tipo = tipo_bool()
        else if T0 == tipo_bool() ∧
                 T1 == tipo_bool() then
            Exp.tipo = tipo_bool()
        else if T0 == tipo_string() ∧
                 T1 == tipo_string() then
            Exp.tipo = tipo_string()
        else
            if Exp == exp_ig(_, _) ∨
               Exp == exp_dist(_, _) then
                tipado_rel_indir(Opnd0, Opnd1, Exp)
            else
                Exp.tipo = error
                aviso_error_bin(T0, T1)
            endif
        endif
    end let

tipado_rel_indir(T0, T1, Exp) :
    if (T0 == tipo_indir() ∨
        T0 == null) ∧
       (T1 == tipo_indir() ∧
        T1 == null) then
        Exp.tipo = tipo_bool()
    else
        Exp.tipo = error
        aviso_error_bin(T0, T1)
    endif

```

3.3. Funciones auxiliares

```

aviso_error_bin(T0, T1) :
  if T0 != error ∧ T1 != error then
    error
  endif

aviso_error_un(T) :
  if T != error then
    error
  endif

ambos_ok(T0, T1) :
  if T0 == ok ∧ T1 == ok then
    return ok
  else
    return error
  endif

ref!(T) :
  if T == tipo_type(_) then
    let T.vinculo = dec_type(Tipo, iden) in
      return ref!(Tipo)
    end let
  else
    return T
  endif

es_designador(E) :
  return E == exp_iden(_) ∨
    E == exp_index(_, _) ∨
    E == exp_reg(_, _) ∨
    E == exp_indir(_)

compatibles(T0, T1) :
  set = nuevoConjunto()
  inserta(set, T0, T1)
  return unificables(T0, T1)

unificables(T0, T1) :
  let T0' = ref!(T0) ∧ T1' = ref!(T1) in
    if T0' == tipo_array(Tipo0, _) ∧ T1' == tipo_array(Tipo1, _) then
      return son_unificables(Tipo0, Tipo1)
    else if T0' == tipo_struct(LCampos0) ∧ T1' == tipo_struct(LCampos1) then
      return son_unificables_struct(LCampos0, LCampos1)
    else if T0' == tipo_indir(_) ∧ T1' == null then
      return true
    else if T0' == tipo_indir(Tipo0) ∧ T1' == tipo_indir(Tipo1) then
      return son_unificables(Tipo0, Tipo1)
    else if T0' == T1' ∨ (T0' == tipo_real() ∧ T1' == tipo_int()) then
      return true
    else
      return false
    endif
  end let

son_unificables_struct(LCampos0, LCampos1) :
  if LCampos0 == un_campo(TipoNom0) ∧ LCampos1 == un_campo(TipoNom1) then
    let TipoNom0 = TipoNom(Tipo0, _) ∧ TipoNom1 = TipoNom(Tipo1, _)
    return son_unificables(Tipo0, Tipo1)
  endif

```

```

        end let
    else if LCampos0 == muchos_campos(LCampos0, TipoNom0) ∧
        LCampos1 == muchos_campos(LCampos1, TipoNom1) then
        let TipoNom0 = TipoNom(Tipo0, _) ∧ TipoNom1 = TipoNom(Tipo1, _)
        return son_unificables(Tipo0, Tipo1) ∧
            son_unificables_struct(LCampos0, LCampos1)
    end let
else
    return false
endif

son_unificables(T0, T1) :
    if contiene(set, T0, T1) then
        return true
    else
        inserta(set, T0, T1)
        return unificables(T0, T1)
    endif

campo_struct(LCampos, iden) :
    if LCampos == un_campo(TipoNom0) then
        let TipoNom0 = TipoNom(Tipo0, iden0)
        if iden == iden0 then
            return Tipo0
        else
            return error
        endif
    end let
    else if LCampos == muchos_campos(LCampos0, TipoNom0) then
        let TipoNom0 = TipoNom(Tipo0, iden0)
        if iden == iden0 then
            return Tipo0
        else
            return campo_struct(LCampos0, iden)
        endif
    end let
endif

num_params(LPParamRs, LPParamFs) :
    if LPParamRs == un_param_r(_) ∧ LPParamFs == un_param_f(_) then
        return true
    else if LPParamRs == muchos_params_r(LPParamRs0, _) ∧
        LPParamFs == muchos_params_f(LPParamFs0, _) then
        return num_params(LPParamRs0, LPParamFs0)
    else
        return false
    endif

compatibles_params(LPParamRs, LPParamFs) :
    if LPParamRs == un_param_r(Exp) ∧ LPParamFs == un_param_f(ParamF) then
        return param_r_f(Exp, ParamF)
    else if LPParamRs == muchos_params_r(LPParamRs0, Exp) ∧
        LPParamFs == muchos_params_f(LPParamFs0, Tipo) then
        return param_r_f(Exp, ParamF) ∧
            compatibles_params(LPParamRs0, LPParamFs0)
    endif

param_r_f(Exp, ParamF) :
    if ParamF == si_refparam_f(Tipo, _) then
        if es_designador(Exp) then

```

```
    if ParamF == tipo_real() then
        if Exp.tipo == tipo_real() then
            return true
        else
            error
            return false
        endif
    else
        if compatibles(Tipo, Exp.tipo) then
            return true
        else
            error
            return false
        endif
    endif
else
    error
    return false
endif
else
    if compatibles(Tipo, Exp.tipo) then
        return true
    else
        error
        return false
    endif
endif
endif
```

4 | Especificación del Procesamiento de Asignación de Espacio

4.1. Funciones de procesamiento

```

var dir = 0
var max_dir = 0
var nivel = 0
var campos

asig_espacio(bloque(SecDecs, SecIs)) :
    var dir_ant = dir
    asig_espacio(SecDecs)
    asig_espacio(SecIs)
    dir = dir_ant

```

4.1.1. Declaraciones

```

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

asig_espacio(no_decs()) : noop

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

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

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

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

asig_espacio1(dec_base(TipoNom)) :
    let TipoNom = TipoNom(Tipo, iden) in
        asig_tam1(Tipo)
        $.dir = dir
        $.nivel = nivel
        inc_dir(Tipo.tam)
    end let

asig_espacio2(dec_base(TipoNom)) :
    let TipoNom = TipoNom(Tipo, iden) in
        asig_tam1(TipoNom)
    end let

asig_espacio1(dec_type(TipoNom)) :
    let TipoNom = TipoNom(Tipo, iden) in
        asig_tam1(Tipo)

```

```

        $.dir = dir
        $.nivel = nivel
        inc_dir(Tipo.tam)
    end let

asig_espacio2(dec_type(TipoNom)) :
    let TipoNom = TipoNom(Tipo, iden) in
        asig_tam1(TipoNom)
    end let

asig_espacio1(dec_proc(iden, ParamFs, Bloq)) :
    var dir_ant = dir
    var max_dir_ant = max_dir
    nivel ++
    $.nivel = nivel
    dir = 0
    max_dir = 0
    asig_espacio1(ParamFs)
    asig_espacio2(ParamFs)
    asig_espacio(Bloq)
    $.tam = max_dir
    dir = dir_ant
    max_dir = max_dir_ant
    nivel --

asig_espacio2(dec_proc(iden, ParamFs, Bloq)) : noop

asig_espacio1(si_params_f(LParamFs)) :
    asig_espacio1(LParamFs)

asig_espacio2(si_params_f(LParamFs)) :
    asig_espacio2(LParamFs)

asig_espacio1(no_params_f()) : noop

asig_espacio2(no_params_f()) : noop

asig_espacio1(muchos_params_f(LParamFs, ParamF)) :
    asig_espacio1(LParamFs)
    asig_espacio1(ParamF)

asig_espacio2(muchos_params_f(LParamFs, ParamF)) :
    asig_espacio2(LParamFs)
    asig_espacio2(ParamF)

asig_espacio1(un_param_f(ParamF)) :
    asig_espacio1(ParamF)

asig_espacio2(un_param_f(ParamF)) :
    asig_espacio2(ParamF)

asig_espacio1(si_refparam_f(Tipo, iden)) :
    asig_tam1(Tipo)
    $.dir = dir
    $.nivel = nivel
    inc_dir(1)

asig_espacio2(si_refparam_f(Tipo, iden)) :
    asig_tam2(Tipo)

```



```

asig_espacio1(no_refparam_f(Tipo, iden)) :
    asig_tam1(Tipo)
    $.dir = dir
    $.nivel = nivel
    inc_dir(Tipo.tam)

```

```

asig_espacio2(no_refparam_f(Tipo, iden)) :
    asig_tam2(Tipo)

```

4.1.2. Tipos

```

asig_tam1(tipo_array(Tipo, litEntero)) :
    asig_tam1(Tipo)
    $.tam = Tipo.tam * litEntero

```

```

asig_tam2(tipo_array(Tipo, litEntero)) : noop

```

```

asig_tam1(tipo_indir(Tipo)) :
    if Tipo != tipo_type(_) then
        asig_tam1(Tipo)
    endif
    $.tam = 1

```

```

asig_tam2(tipo_indir(Tipo)) :
    if Tipo == tipo_type(iden) then
        let Tipo.vinculo = dec_type(TipoNom)  $\wedge$  TipoNom = TipoNom(T, iden) in
            Tipo.tam = T.tam
        end let
    else
        asig_tam2(Tipo)
    endif

```

```

asig_tam1(tipo_struct(LCampos)) :
    campos = 0
    asig_tam1(LCampos)
    $.tam = campos

```

```

asig_tam2(tipo_struct(LCampos)) :
    asig_tam2(LCampos)

```

```

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

```

```

asig_tam2(tipo_int()) : noop

```

```

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

```

```

asig_tam2(tipo_real()) : noop

```

```

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

```

```

asig_tam2(tipo_bool()) : noop

```

```

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

```

```

asig_tam2(tipo_string()) : noop

```

```

asig_tam1(tipo_type(iden)) :
  let $.vinculo = dec_type(TipoNom)  $\wedge$  TipoNom = TipoNom(T, iden) in
    $.tam = T.tam
  end let

```

```

asig_tam2(tipo_type(iden)) : noop

```

```

asig_tam1(muchos_campos(LCampos, TipoNom)) :
  asig_tam1(LCampos)
  let TipoNom = TipoNom(T, iden) in
    asig_tam1(T)
    campos += T.tam
  end let

```

```

asig_tam2(muchos_campos(LCampos, TipoNom)) :
  asig_tam2(LCampos)
  let TipoNom = TipoNom(T, iden) in
    asig_tam2(T)
  end let

```

```

asig_tam1(un_campo(TipoNom)) :
  let TipoNom = TipoNom(T, iden) in
    asig_tam1(T)
    campos += T.tam
  end let

```

```

asig_tam2(un_campo(TipoNom)) :
  let TipoNom = TipoNom(T, iden) in
    asig_tam2(T)
  end let

```

4.1.3. Instrucciones

```

asig_espacio(si_ins(LIs)) :
  asig_espacio(LIs)

```

```

asig_espacio(no_ins()) : noop

```

```

asig_espacio(muchas_ins(LIs, I)) :
  asig_espacio(LIs)
  asig_espacio(I)

```

```

asig_espacio(una_ins(I)) :
  asig_espacio(I)

```

```

asig_espacio(ins_eval(Exp)) : noop

```

```

asig_espacio(ins_if(Exp, Bloq)) :
  asig_espacio(Bloq)

```

```

asig_espacio(ins_if_else(I, Bloq)) :
  asig_espacio(I)
  asig_espacio(Bloq)

```

```

asig_espacio(ins_while(Exp, Bloq)) :
  asig_espacio(Bloq)

```

```

asig_espacio(ins_read(Exp)) : noop

```

```

asig_espacio(ins_write(Exp)) : noop

```

```
asig_espacio(ins_nl()) : noop  
asig_espacio(ins_new(Exp)) : noop  
asig_espacio(ins_delete(Exp)) : noop  
asig_espacio(ins_call(iden, ParamRs)) : noop  
asig_espacio(ins_bloque(Bloq)) :  
    asig_espacio(Bloq)
```

4.2. Funciones auxiliares

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

5 | Descripción de las Instrucciones de la Máquina-p

Se asume que el elemento en la cima de la pila es el `elem1`, y aquel que está debajo el `elem2`, por tanto, en el fondo de la pila se encuentra el `elemN`, siendo `N` el número de elementos en la pila

5.1. Operaciones

- `suma()`: Desapila los dos valores en la cima de la pila, y apila `elem2 + elem1`
- `resta()`: Desapila los dos valores en la cima de la pila, y apila `elem2 - elem1`
- `mul()`: Desapila los dos valores en la cima de la pila, y apila `elem2 * elem1`
- `div()`: Desapila los dos valores en la cima de la pila, y apila `elem2 / elem1`
- `and()`: Desapila los dos valores en la cima de la pila, y apila `elem2 && elem1`
- `or()`: Desapila los dos valores en la cima de la pila, y apila `elem2 || elem1`
- `menos()`: Desapila la cima de la pila, y apila `-(elem1)`
- `not()`: Desapila la cima de la pila, y apila `!(elem1)`
- `menor()`: Desapila los dos valores en la cima de la pila, y apila `elem2 < elem1`
- `menor_ig()`: Desapila los dos valores en la cima de la pila, y apila `elem2 <= elem1`
- `mayor()`: Desapila los dos valores en la cima de la pila, y apila `elem2 > elem1`
- `mayor_ig()`: Desapila los dos valores en la cima de la pila, y apila `elem2 >= elem1`
- `ig()`: Desapila los dos valores en la cima de la pila, y apila `elem2 == elem1`
- `dist()`: Desapila los dos valores en la cima de la pila, y apila `elem2 != elem1`

5.2. Tipos básicos

- `apila_int(n)`: Apila el valor del entero `n`
- `apila_real(r)`: Apila el valor de real `r`
- `apila_bool(b)`: Apila el valor del booleano `b`
- `apila_string(s)`: Apila el valor del string `s`
- `apila_null()`: Apila el valor null

5.3. Direccionamiento

- `apila_ind()`: Desapila la cima de la pila, y apila el contenido de la memoria en la celda `elem1`
- `desapila_ind()`: Desapila los dos valores en la cima de la pila, y almacena el valor `elem1` en la memoria, en la celda `elem2`
- `copia(N)`: Desapila los dos valores en la cima de la pila (siendo `elem1` la celda donde se encuentra el primer elemento origen, y `elem2` la celda donde se encuentra el primer elemento destino), y almacena sendos elementos del origen en el destino hasta haber copiado `N` elementos
- `alloc(tam)`: Reserva un espacio en la memoria dinámica de tamaño `tam`, y apila la dirección de inicio del espacio reservado

- `dealloc(tam)`: Desapila la cima de la pila, y libera el espacio en la memoria dinámica de tamaño `tam`, que tiene como dirección de inicio `elem1`

5.4. Procedimientos

- `activa(n, t, d)`:
- `desactiva(n, t)`:
- `dup()`: Desapila la cima de la pila, y apila dos veces `elem1`

5.5. Anidamiento

- `apilad(n)`: Apila el valor del display del nivel `n`
- `desapilad(n)`: Desapila la cima de la pila, y la introduce en el display de nivel `n`

5.6. Saltos

- `ir_a(dir)`: Cambia el valor del PC a `dir`
- `ir_f(dir)`: Desapila la cima de la pila, y si el valor `elem0` es falso cambia el valor del PC a `dir`
- `ir_ind()`: Desapila la cima de la pila, y cambia el valor del PC a `elem1`

5.7. I/O

- `entrada_std(exp)`: Apila el respectivo valor recibido en la pila
- `salida_std()`: Desapila la cima de la pila, y vuelca en la consola el valor de `elem1`
- `nl()`: Vuelca un salto de línea sobre la consola

5.8. Otros

- `desapila()`: Desapila la cima de la pila
- `stop()`: Detiene la máquina

6 | Especificación del Procesamiento de Etiquetado

6.1. Funciones para la pila de procedimientos

- `nuevaPila()`: Crea una pila vacía para almacenar los procedimientos
- `pilaVacía(stack)`: Comprueba si la pila `stack` contiene algún elemento.
- `cima(stack)`: Devuelve el procedimiento en la cima de la pila `stack`.
- `desapila(stack)`: Desapila el procedimiento en la cima de la pila `stack`.
- `apila(stack, proc)`: Apila el procedimiento `proc` en la cima de la pila `stack`.

6.2. Funciones de procesamiento

```

var sub_pendientes = nuevaPila()
var etq = 0

etiquetado(bloque(SecDecs, SecIs)) :
    $.prim = etq
    recolecta_subs(SecDecs)
    etiquetado(SecIs)
    if bloque.programa then
        etq ++
        while !pilaVacía(sub_pendientes)
            sub = cima(sub_pendientes)
            desapila(sub_pendientes)
            let sub = dec_proc(iden, ParamFs, Bloq) in
                sub.prim = etq
                etq ++
                etiq(Bloq)
                etq += 2
                sub.sig = etq
            end let
        end while
    endif
    $.sig = etq

```

6.2.1. Declaraciones

```

recolecta_subs(si_decs(LDecs)) :
    recolecta_subs(LDecs)

recolecta_subs(no_decs()) : noop

recolecta_subs(muchas_decs(LDecs, Dec)) :
    recolecta_subs(LDecs)
    recolecta_subs(Dec)

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

```

recolecta_sub(dec_base(*TipoNom*)) : **noop**

recolecta_sub(dec_type(*TipoNom*)) : **noop**

recolecta_sub(dec_proc(*iden*, *ParamFs*, *Bloq*)) :
apila(sub_pendientes, \$)

6.2.2. Instrucciones

etiquetado(si_ins(*LIs*)) :
 \$.prim = etq
etiquetado(*LIs*)
 \$.sig = etq

etiquetado(no_ins()) : **noop**

etiquetado(muchas_ins(*LIs*, *I*)) :
 \$.prim = etq
etiquetado(*LIs*)
etiquetado(*I*)
 \$.sig = etq

etiquetado(una_ins(*I*)) :
 \$.prim = etq
etiquetado(*I*)
 \$.sig = etq

etiquetado(ins_eval(*Exp*)) :
 \$.prim = etq
etiquetado(*Exp*)
 if *Exp* != exp_asig(*Exp*₀, *Exp*₁)
 etq ++
 endif
 \$.sig = etq

etiquetado(ins_if(*Exp*, *Bloq*)) :
 \$.prim = etq
etiquetado(*Exp*)
etiquetado_acc_val(*Exp*)
 etq ++
etiquetado(*Bloq*)
 \$.sig = etq

etiquetado(ins_if_else(*I*, *Bloq*)) :
 \$.prim = etq
etiquetado(*I*)
 etq ++
etiquetado(*Bloq*)
 \$.sig = etq

etiquetado(ins_while(*Exp*, *Bloq*)) :
 \$.prim = etq
etiquetado(*Exp*)
etiquetado_acc_val(*Exp*)
 etq ++
etiquetado(*Bloq*)
 etq ++
 \$.sig = etq

etiquetado(ins_read(*Exp*)) :

```

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

```

```

etiquetado(ins_write(Exp)) :
$.prim = etq
etiquetado(Exp)
etiquetado_acc_val(Exp)
etq ++
$.sig = etq

```

```

etiquetado(ins_nl()) :
$.prim = etq
etq ++
$.sig = etq

```

```

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

```

```

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

```

```

etiquetado(ins_call(iden, ParamRs)) :
$.prim = etq
let $.vinculo = dec_proc(iden, ParamFs, Bloq) in
    etq ++
    if ParamRs = si_params_r(LParamRs) ∧ ParamFs = si_params_f(LParamFs) then
        etiquetado_paso_params(LParamRs, LParamFs)
    endif
    etq ++
end let
$.sig = etq

```

```

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

```

6.2.3. Expresiones

```

etiquetado(exp_asig(Opnd0, Opnd1)) :
$.prim = etq
etiquetado(Opnd0)
etiquetado(Opnd1)
etq ++
$.sig = etq

```

```

etiquetado(exp_menor(Opnd0, Opnd1)) :
$.prim = etq
etiquetado_opnds(Opnd0, Opnd1)
etq ++
$.sig = etq

```



```

etiquetado(exp_menor_ig(Opnd0, Opnd1)) :
    $.prim = etq
    etiquetado_opnds(Opnd0, Opnd1)
    etq++
    $.sig = etq

```

```

etiquetado(exp_mayor(Opnd0, Opnd1)) :
    $.prim = etq
    etiquetado_opnds(Opnd0, Opnd1)
    etq++
    $.sig = etq

```

```

etiquetado(exp_mayor_ig(Opnd0, Opnd1)) :
    $.prim = etq
    etiquetado_opnds(Opnd0, Opnd1)
    etq++
    $.sig = etq

```

```

etiquetado(exp_ig(Opnd0, Opnd1)) :
    $.prim = etq
    etiquetado_opnds(Opnd0, Opnd1)
    etq++
    $.sig = etq

```

```

etiquetado(exp_dist(Opnd0, Opnd1)) :
    $.prim = etq
    etiquetado_opnds(Opnd0, Opnd1)
    etq++
    $.sig = etq

```

```

etiquetado(exp_suma(Opnd0, Opnd1)) :
    $.prim = etq
    etiquetado_opnds(Opnd0, Opnd1)
    etq++
    $.sig = etq

```

```

etiquetado(exp_resta(Opnd0, Opnd1)) :
    $.prim = etq
    etiquetado_opnds(Opnd0, Opnd1)
    etq++
    $.sig = etq

```

```

etiquetado(exp_and(Opnd0, Opnd1)) :
    $.prim = etq
    etiquetado_opnds(Opnd0, Opnd1)
    etq++
    $.sig = etq

```

```

etiquetado(exp_or(Opnd0, Opnd1)) :
    $.prim = etq
    etiquetado_opnds(Opnd0, Opnd1)
    etq++
    $.sig = etq

```

```

etiquetado(exp_mul(Opnd0, Opnd1)) :
    $.prim = etq
    etiquetado_opnds(Opnd0, Opnd1)
    etq++
    $.sig = etq

```

```

etiquetado(exp_div(Opnd0, Opnd1)) :
    $.prim = etq
    etiquetado_opnds(Opnd0, Opnd1)
    etq++
    $.sig = etq

```

```

etiquetado(exp_mod(Opnd0, Opnd1)) :
    $.prim = etq
    etiquetado_opnds(Opnd0, Opnd1)
    etq++
    $.sig = etq

```

```

etiquetado(exp_menos(Opnd)) :
    $.prim = etq
    etiquetado(Opnd)
    etiquetado_acc_val(Opnd)
    etq++
    $.sig = etq

```

```

etiquetado(exp_not(Opnd)) :
    $.prim = etq
    etiquetado(Opnd)
    etiquetado_acc_val(Opnd)
    etq++
    $.sig = etq

```

```

etiquetado(exp_index(Opnd0, Opnd1)) :
    $.prim = etq
    etiquetado(Opnd0)
    etiquetado(Opnd1)
    etiquetado_acc_val(Opnd1)
    etq += 3
    $.sig = etq

```

```

etiquetado(exp_reg(Opnd, iden)) :
    $.prim = etq
    etiquetado(Opnd)
    etq += 2
    $.sig = etq

```

```

etiquetado(exp_indir(Opnd)) :
    $.prim = etq
    etiquetado(Opnd)
    etq++
    $.sig = etq

```

```

etiquetado(exp_entero(litEntero)) :
    $.prim = etq
    etq++
    $.sig = etq

```

```

etiquetado(exp_real(litReal)) :
    $.prim = etq
    etq++
    $.sig = etq

```

```

etiquetado(exp_true()) :
    $.prim = etq
    etq++
    $.sig = etq

```

```

etiquetado(exp_false()) :
    $.prim = etq
    etq ++
    $.sig = etq

etiquetado(exp_cadena(litCadena)) :
    $.prim = etq
    etq ++
    $.sig = etq

etiquetado(exp_iden(iden)) :
    $.prim = etq
    etiquetado_acc_id(iden.vinculo)
    $.sig = etq

etiquetado(exp_null()) :
    $.prim = etq
    etq ++
    $.sig = etq

```

6.3. Funciones auxiliares

```

etiquetado_acc_val(Exp) :
    if es_designador(Exp) then
        etq ++
    endif

etiquetado_acc_id(dec_base(TipoNom)) :
    if dec_base.nivel == 0 then
        etq ++
    else
        etq += 3
    endif

etiquetado_acc_id(no_refparam_f(Tipo, iden)) :
    etq += 3

etiquetado_acc_id(si_refparam_f(Tipo, iden)) :
    etq += 4

etiquetado_opnds(Opnd0, Opnd1) :
    etiquetado(Opnd0)
    etiquetado_acc_val(Opnd0)
    etiquetado(Opnd1)
    etiquetado_acc_val(Opnd1)

etiquetado_paso_params(LParamRs, LParamFs) :
    if LParamRs == un_param_r(Exp) ∧ LParamFs == un_param_f(ParamF) then
        param_r_f(Exp, ParamF)
    else if LParamRs == muchos_params_r(LParamRs0, Exp) ∧
        LParamFs == muchos_params_f(LParamFs0, Tipo) then
        param_r_f(Exp, ParamF)
        gen_paso_params(LParamRs0 LParamFs0)
    endif

param_r_f(Exp, ParamF) :
    etq += 3
    etiquetado(Exp)

```

etq ++

7 | Especificación del Procesamiento de Generación de Código

7.1. Funciones para la pila de procedimientos

- `nuevaPila()`: Crea una pila vacía para almacenar los procedimientos
- `pilaVacía(stack)`: Comprueba si la pila `stack` contiene algún elemento.
- `cima(stack)`: Devuelve el procedimiento en la cima de la pila `stack`.
- `desapila(stack)`: Desapila el procedimiento en la cima de la pila `stack`.
- `apila(stack, proc)`: Apila el procedimiento `proc` en la cima de la pila `stack`.

7.2. Funciones de procesamiento

```

var sub_pendientes = nuevaPila()
gen_cod(bloque(SecDecs, SecIs)) :
  recolecta_subs(SecDecs)
  gen_cod(SecIs)
  emit_stop()
  if bloque.programa then
    while !pilaVacía(sub_pendientes)
      sub = cima(sub_pendientes)
      desapila(sub_pendientes)
      let sub = dec_proc(iden, ParamFs, Bloq) in
        emit_desapilad(sub.nivel)
        gen_cod(Bloq)
        emit_desactiva(sub.nivel, sub.tam)
        emit_ir_ind()
      end let
    end while
  endif

```

7.2.1. Declaraciones

```

recolecta_subs(si_decs(LDecs)) :
  recolecta_subs(LDecs)

recolecta_subs(no_decs()) : noop

recolecta_subs(muchas_decs(LDecs, Dec)) :
  recolecta_subs(LDecs)
  recolecta_subs(Dec)

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

recolecta_subs(dec_base(TipoNom)) : noop

recolecta_subs(dec_type(TipoNom)) : noop

```

```
recolecta_subs(dec_proc(iden, ParamFs, Bloq)) :
  apila(sub_pendientes, $)
```

7.2.2. Instrucciones

```
gen_cod(si_ins(LIs)) :
  gen_cod(LIs)
```

```
etiquetado(no_ins()) : noop
```

```
gen_cod(muchas_ins(LIs, I)) :
  gen_cod(LIs)
  gen_cod(I)
```

```
gen_cod(una_ins(I)) :
  gen_cod(I)
```

```
gen_cod(ins_eval(Exp)) :
  gen_cod(Exp)
  if Exp != exp_asig(Exp0, Exp1)
    emit desapila()
  endif
```

```
gen_cod(ins_if(Exp, Bloq)) :
  gen_cod(Exp)
  gen_cod_acc_val(Exp)
  emit ir_f($.sig)
  gen_cod(Bloq)
```

```
gen_cod(ins_if_else(I, Bloq)) :
  let I = ins_if(Exp, Bloq0) in
    gen_cod(Exp)
    gen_cod_acc_val(Exp)
    emit ir_f(I.sig + 1)
    gen_cod(Bloq)
  end let
  emit ir_a($.sig)
  gen_cod(Bloq)
```

```
gen_cod(ins_while(Exp, Bloq)) :
  gen_cod(Exp)
  gen_cod_acc_val(Exp)
  emit ir_f($.sig)
  gen_cod(Bloq)
  emit ir_a($.prim)
```

```
gen_cod(ins_read(Exp)) :
  gen_cod(Exp)
  emit entrada_std(Exp)
  emit desapila_ind()
```

```
gen_cod(ins_write(Exp)) :
  gen_cod(Exp)
  gen_cod_acc_val(Exp)
  emit salida_std()
```

```
gen_cod(ins_nl()) :
  emit nl()
```

```
gen_cod(ins_new(Exp)) :
```

```

    gen_cod(Exp)
    let ref!(Exp.tipo) = tipo_indir(Tipo) in
        emit alloc(Tipo.tam)
    end let
emit desapila_ind()

gen_cod(ins_delete(Exp)) :
    gen_cod(Exp)
emit apila_ind()
    let ref!(Exp.tipo) = tipo_indir(Tipo) in
        emit dealloc(Tipo.tam)
    end let

gen_cod(ins_call(iden, ParamRs)) :
    let $.vinculo = dec_proc(iden, ParamFs, Bloq) in
        emit activa($.vinculo.nivel, $.vinculo.tam, $.sig)
        if ParamRs = si_params_r(LParamRs)  $\wedge$  ParamFs = si_params_f(LParamFs) then
            gen_paso_params(LParamRs, LParamFs)
        endif
        emit ir_a($.vinculo.prim)
    end let

gen_cod(ins_bloque(Bloq)) :
    gen_cod(Bloq)

```

7.2.3. Expresiones

```

gen_cod(exp_asig(Opnd0, Opnd1)) :
    gen_cod(Opnd0)
    gen_cod(Opnd1)
    if es_designador(Opnd1) then
        emit copia(Opnd1.tipo.tam)
    else
        emit desapila_ind()
    endif

gen_cod(exp_menor(Opnd0, Opnd1)) :
    gen_cod_opnds(Opnd0, Opnd1)
    emit menor()

gen_cod(exp_menor_ig(Opnd0, Opnd1)) :
    gen_cod_opnds(Opnd0, Opnd1)
    emit menor_ig()

gen_cod(exp_mayor(Opnd0, Opnd1)) :
    gen_cod_opnds(Opnd0, Opnd1)
    emit mayor()

gen_cod(exp_mayor_ig(Opnd0, Opnd1)) :
    gen_cod_opnds(Opnd0, Opnd1)
    emit mayor_ig()

gen_cod(exp_ig(Opnd0, Opnd1)) :
    gen_cod_opnds(Opnd0, Opnd1)
    emit ig()

gen_cod(exp_dist(Opnd0, Opnd1)) :
    gen_cod_opnds(Opnd0, Opnd1)
    emit dist()

```

```

gen_cod(exp_suma(Opnd0, Opnd1)) :
    gen_cod_opnds(Opnd0, Opnd1)
    emit suma()

gen_cod(exp_resta(Opnd0, Opnd1)) :
    gen_cod_opnds(Opnd0, Opnd1)
    emit resta()

gen_cod(exp_and(Opnd0, Opnd1)) :
    gen_cod_opnds(Opnd0, Opnd1)
    emit and()

gen_cod(exp_or(Opnd0, Opnd1)) :
    gen_cod_opnds(Opnd0, Opnd1)
    emit or()

gen_cod(exp_mul(Opnd0, Opnd1)) :
    gen_cod_opnds(Opnd0, Opnd1)
    emit mul()

gen_cod(exp_div(Opnd0, Opnd1)) :
    gen_cod_opnds(Opnd0, Opnd1)
    emit div()

gen_cod(exp_mod(Opnd0, Opnd1)) :
    gen_cod_opnds(Opnd0, Opnd1)
    emit mod()

gen_cod(exp_menos(Opnd)) :
    gen_cod(Opnd)
    gen_cod_acc_val(Opnd)
    emit menos()

gen_cod(exp_not(Opnd)) :
    gen_cod(Opnd)
    gen_cod_acc_val(Opnd)
    emit not()

gen_cod(exp_index(Opnd0, Opnd1)) :
    gen_cod(Opnd0)
    gen_cod(Opnd1)
    gen_acc_val(Opnd1)
    let ref!(Opnd0) = array(Tipo, iden) in
        emit apila_int(Tipo.tam)
    end let
    emit mul()
    emit suma()

gen_cod(exp_reg(Opnd, iden)) :
    gen_cod(Opnd)
    let ref!(Opnd) = tipo_struct(LCampos) in
        emit apila_int(desplaza_campo(LCampos, iden))
    end let
    emit suma()

gen_cod(exp_indir(Opnd)) :
    gen_cod(Opnd)
    apila_ind()

gen_cod(exp_entero(litEntero)) :

```



```

    emit apila_int(litEntero)

gen_cod(exp_real(litReal)) :
    emit apila_real(litReal)

gen_cod(exp_true()) :
    emit apila_bool(true)

gen_cod(exp_false()) :
    emit apila_bool(false)

gen_cod(exp_cadena(litCadena)) :
    emit apila_string(litCadena)

gen_cod(exp_iden(iden)) :
    gen_cod_acc_id(iden.vinculo)

gen_cod(exp_null()) :
    emit apila_null()

```

7.3. Funciones auxiliares

```

gen_acc_val(Exp) :
    if es_designador(Exp) then
        emit apila_ind()
    endif

gen_cod_acc_id(dec_base(TipoNom)) :
    if dec_base.nivel == 0 then
        emit apila_int(dec_base.dir)
    else
        gen_acc_var(dec_base)
    endif

gen_cod_acc_id(no_refparam_f(Tipo, iden)) :
    gen_acc_var(no_refparam_f)

gen_cod_acc_id(si_refparam_f(Tipo, iden)) :
    gen_acc_var(si_refparam_f)
    emit apila_ind()

gen_acc_var(V) :
    emit apilad(V.nivel)
    emit apila_int(v.dir)
    emit suma()

gen_opnds(Opnd0, Opnd1) :
    gen_cod(Opnd0)
    gen_acc_val(Opnd0)
    gen_cod(Opnd1)
    gen_acc_val(Opnd1)

gen_paso_params(LParamRs, LParamFs) :
    if LParamRs == un_param_r(Exp) ∧ LParamFs == un_param_f(ParamF) then
        param_r_f(Exp, ParamF)
    else if LParamRs == muchos_params_r(LParamRs0, Exp) ∧
        LParamFs == muchos_params_f(LParamFs0, Tipo) then
        param_r_f(Exp, ParamF)
        gen_paso_params(LParamRs0 LParamFs0)
    end

```

endif

```

param_r_f(Exp, ParamF) :
  emit dup()
  emit apila_int(ParamF.dir)
  emit suma()
  gen_cod(Exp)
  if ParamF == si_refparam_f(_, _)  $\vee$  !les_designador(Exp) then
    emit desapila_ind()
  else if ParamF == no_refparam_f(Tipo, _)
    emit copia(Tipo.tam)
  endif

desplaza_campo(LCampos, iden) :
  if LCampos == un_campo(TipoNom0) then
    let TipoNom0 = TipoNom(Tipo0, iden0)
    return 1
  end let
  else if LCampos == muchos_campos(LCampos0, TipoNom0) then
    let TipoNom0 = TipoNom(Tipo0, iden0)
    if iden == iden0 then
      return 1
    else
      return desplaza_campo(LCampos0, iden) + Tipo0.tam
    endif
  end let
endif

```