```
with Ada.Sequential_IO;
with d_pila;
with decls; use decls;
with decls.d_tnoms; use decls.d_tnoms;
with decls.d_tsimbols; use decls.d_tsimbols;
with decls.d_arbre; use decls.d_arbre;
with decls.d_c3a; use decls.d_c3a;
package semantica is

  procedure prepara_analisi(nomf: in String);

private

  tn: tnoms;
  ts: tsimbols;

  root: pnode;

  nv: num_var;
  np: num_proc;
  ne: num_etiq;

  tv: tvariables;
  tp: tprocediments;

  ERROR: boolean;

  package Instruccio_IO is new Ada.Sequential_IO(instr_3a_bin);
  package Pila_Procediments is new D_Pila(max_proc, elem => num_proc);

  pproc: Pila_Procediments.pila;

end semantica;



--BODY
with semantica.missatges;
with semantica.c_tipus;
with semantica.g_codi_int;
with semantica.g_codi_ass;
package body semantica is

  procedure prepara_analisi(nomf: in String) is
    cert, fals: num_var;
  begin

    empty(tn);
    empty(ts);

    nv:= null_nv;
    np:= null_np;
    ne:= null_ne;

    ERROR:= false;

    semantica.c_tipus.posa_entorn_standard(cert, fals);
    semantica.g_codi_int.prepara_g_codi_int(nomf, cert, fals);
    semantica.g_codi_ass.prepara_g_codi_ass(nomf);
  end prepara_analisi;

end semantica;
```

```
with decls.d_descripcio;
package semantica.c_arbre is

   -- Rutines de control
   procedure rl_atom(a: out atribut);
   procedure rs_atom(a: out atribut);

   -- Rutines Fonamentals
   procedure rl_identifier(a: out atribut; pos: in posicio; text: in String);
   procedure rl_literal_ent(a: out atribut; pos: in posicio; text: in String);
   procedure rl_literal_car(a: out atribut; pos: in posicio; text: in String);
   procedure rl_literal_str(a: out atribut; pos: in posicio; text: in String);

   -- Operadors relacionals
   procedure rl_op_menor(a: out atribut);
   procedure rl_op_major(a: out atribut);
   procedure rl_op_menorigual(a: out atribut);
   procedure rl_op_majorigual(a: out atribut);
   procedure rl_op_igual(a: out atribut);
   procedure rl_op_diferent(a: out atribut);

   -- Procediment
   procedure rs_Root(proc: in atribut);
   procedure rs_Proc(proc: out atribut; cproc: in atribut; decls: in atribut;
                     sents: in atribut);
   procedure rs_C_Proc(cproc: out atribut; proc_id: in atribut;  args: in atribut);
   procedure rs_C_Proc(cproc: out atribut; proc_id: in atribut);
   procedure rs_Args(args: out atribut; args_seg: in atribut; arg: in atribut);
   procedure rs_Args(args: out atribut; arg: in atribut);
   procedure rs_Arg(arg: out atribut; lid: in atribut; mode: in atribut; tipus: in atribut);
   procedure rs_Mode_in(mode: out atribut);
   procedure rs_Mode_in_out(mode: out atribut);

   -- Declaracio
   procedure rs_Decls(decls: out atribut; decls_seg: in atribut; decl: in atribut);
   procedure rs_Decl(decl: out atribut; decl_real: in atribut);
   procedure rs_Decl_Var(decl: out atribut; lista_id: in atribut; tipus: in atribut);
   procedure rs_Decl_Const(decl: out atribut; lid_const: in atribut;
                           tipus: in atribut; valor: in atribut);
   procedure rs_Idx_neg(idx: out atribut; idx_cont: in atribut);
   procedure rs_Idx_pos(idx: out atribut; idx_cont: in atribut);
   procedure rs_Idx_Cont(idx_cont: out atribut; valor: in atribut);
   procedure rs_Decl_T(decl: out atribut; id_type: in atribut; decl_cont: in atribut);
   procedure rs_Decl_T_Cont(decl: out atribut; info: in atribut);
   procedure rs_Decl_T_Cont(decl: out atribut; rang_array: in atribut;
                            tipus_array: in atribut);
   procedure rs_DCamps(camps: out atribut; camp_seg: in atribut; camp: in atribut);
   procedure rs_DCamps(camps: out atribut; camp: in atribut);
   procedure rs_DCamp(camp: out atribut; var: in atribut);
   procedure rs_Rang(rang: out atribut; id_type: in atribut; linf: in atribut;
                     lsup: in atribut);

   -- Sentencia
   procedure rs_Sents(sents: out atribut; sent: in atribut);
   procedure rs_Sent_Nob(sents: out atribut; sent_cont: in atribut; sent: in atribut);
   procedure rs_Sent_Nob(sents: out atribut; sent: in atribut);
   procedure rs_Sent(sent: out atribut; stipus: in atribut);
   procedure rs_SIter(sent: out atribut; expr: in atribut; sents: in atribut);
   procedure rs_SCond(sent: out atribut; expr: in atribut; sents: in atribut);
   procedure rs_SCond(sent: out atribut; expr: in atribut;
                      sents_if: in atribut; sents_else: in atribut);
   procedure rs_SCrida(sent: out atribut; ref: in atribut);
   procedure rs_SAssign(sent: out atribut; ref: in atribut; expr: in atribut);

   -- Expressio
   procedure rs_LExpr(lexpr: out atribut; cont: in atribut; expr: in atribut);
   procedure rs_LExpr(lexpr: out atribut; expr: in atribut);
   procedure rs_Expr(expr: out atribut; cont: in atribut);
   procedure rs_EAnd(expr: out atribut; ee: in atribut; ed: in atribut);
   procedure rs_EOr(expr: out atribut; ee: in atribut; ed: in atribut);
   procedure rs_EOpo(expr: out atribut; ee: in atribut; ed: in atribut; op: in atribut);
   procedure rs_EOps(expr: out atribut; ee: in atribut; ed: in atribut);
   procedure rs_EOpr(expr: out atribut; ee: in atribut; ed: in atribut);
   procedure rs_EOpp(expr: out atribut; ee: in atribut; ed: in atribut);
   procedure rs_EOpq(expr: out atribut; ee: in atribut; ed: in atribut);
   procedure rs_EOpm(expr: out atribut; ee: in atribut; ed: in atribut);
   procedure rs_EOpnl(expr: out atribut; ed: in atribut);
   procedure rs_EOpna(expr: out atribut; ed: in atribut);
   procedure rs_EOp(expr: out atribut; ed: in atribut);
   procedure rs_ET(expr: out atribut; e: in atribut);
```

```
  -- Referència
  procedure rs_Ref(ref: out atribut; ref_id: in atribut; qs: in atribut);
  procedure rs_Qs(qs: out atribut; qs_in: in atribut; q: in atribut);
  procedure rs_Q(q: out atribut; contingut: in atribut);


  -- Llistes
  procedure rs_Lid(lid: out atribut; id_seg: in atribut; id: in atribut);
  procedure rs_Lid(lid: out atribut; id: in atribut);


  err,type_error,proc_error,arg_error,record_error,camp_error,array_error,
  var_error,const_error: exception;

end semantica.c_arbre;



--BODY
with decls.d_descripcio;
with decls.d_arbre;
package body semantica.c_arbre is

  -- Rutines de control

  procedure rl_atom(a: out atribut) is
  begin
    a:= new node(nd_null);
  end rl_atom;


  procedure rs_atom(a: out atribut) is
  begin
    a:= new node(nd_null);
  end rs_atom;

  -- Rutines lèxiques

  procedure rl_identifier(a: out atribut; pos: in posicio; text: in String)
  is
    id: id_nom;
  begin
    put(tn,text,id);
    a:= new node(nd_id);
    a.id_pos:= pos; a.id_id:= id;
  end rl_identifier;


  procedure rl_literal_ent(a: out atribut; pos: in posicio; text: in String)
  is
  begin
    a:= new node(nd_lit);
    a.lit_pos:= pos;
    a.lit_val:= valor'Value(text);
    a.lit_tipus:= decls.d_descripcio.tsb_ent;
  end rl_literal_ent;


  procedure rl_literal_car(a: out atribut; pos: in posicio; text: in String)
  is
  begin
    a:= new node(nd_lit);
    a.lit_pos:= pos;
    a.lit_val:= valor(Character'Pos(text(text'First+1)));
    a.lit_tipus:= decls.d_descripcio.tsb_car;
  end rl_literal_car;


  procedure rl_literal_str(a: out atribut; pos: in posicio; text: in String)
  is
    ids: id_str;
  begin
    put(tn,text,ids);
    a:= new node(nd_lit);
    a.lit_pos:= pos;
    a.lit_val:= valor(ids);
    a.lit_tipus:= decls.d_descripcio.tsb_nul;
  end rl_literal_str;
```

```
procedure rl_op_menor(a: out atribut) is
begin
  a:= new node(nd_op_rel);
  a.orel_tipus:= menor;
end rl_op_menor;


procedure rl_op_major(a: out atribut) is
begin
  a:= new node(nd_op_rel);
  a.orel_tipus:= major;
end rl_op_major;


procedure rl_op_menorigual(a: out atribut) is
begin
  a:= new node(nd_op_rel);
  a.orel_tipus:= menorigual;
end rl_op_menorigual;


procedure rl_op_majorigual(a: out atribut) is
begin
  a:= new node(nd_op_rel);
  a.orel_tipus:= majorigual;
end rl_op_majorigual;


procedure rl_op_igual(a: out atribut) is
begin
  a:= new node(nd_op_rel);
  a.orel_tipus:= igual;
end rl_op_igual;


procedure rl_op_diferent(a: out atribut) is
begin
  a:= new node(nd_op_rel);
  a.orel_tipus:= diferent;
end rl_op_diferent;


-- Rutines sintàctiques

procedure rs_Root(proc: in atribut) is
begin
  root:= new node(nd_root);
  root.p:=proc;
end rs_Root;


procedure rs_Proc(proc: out atribut; cproc: in atribut; decls: in atribut;
                  sents: in atribut) is
begin
  proc:= new node(nd_proc);
  proc.proc_cproc:= cproc;
  proc.proc_decls:= decls;
  proc.proc_sents:= sents;
end rs_Proc;


procedure rs_C_Proc(cproc: out atribut; proc_id: in atribut; args: in atribut)
is
begin
  cproc:= new node(nd_c_proc);
  cproc.cproc_id:= proc_id;
  cproc.cproc_args:= args;
  cproc.cproc_np:= null_np;
end rs_C_Proc;


procedure rs_C_Proc(cproc: out atribut; proc_id: in atribut) is
begin
  cproc:= new node(nd_c_proc);
  cproc.cproc_id:= proc_id;
  cproc.cproc_args:= new node(nd_null);
  cproc.cproc_np:= null_np;
end rs_C_Proc;
```

```
procedure rs_Args(args: out atribut; args_seg: in atribut; arg: in atribut)
is
begin
  args:= new node(nd_args);
  args.args_args:= args_seg;
  args.args_arg:= arg;
end rs_Args;


procedure rs_Args(args: out atribut; arg: in atribut) is
begin
  args:= new node(nd_args);
  args.args_args:= new node(nd_null);
  args.args_arg:= arg;
end rs_Args;


procedure rs_Arg(arg: out atribut; lid: in atribut; mode: in atribut; tipus: in atribut) is
begin
  arg:= new node(nd_arg);
  arg.arg_lid:= lid;
  arg.arg_mode:= mode.mode_tipus;
  arg.arg_tipus:= tipus;
end rs_Arg;


procedure rs_Mode_in(mode: out atribut) is
begin
  mode:= new node(nd_mode);
  mode.mode_tipus:= md_in;
end rs_Mode_in;


procedure rs_Mode_in_out(mode: out atribut) is
begin
  mode:= new node(nd_mode);
  mode.mode_tipus:= md_in_out;
end rs_Mode_in_out;


-- Declaracions
procedure rs_Decls(decls: out atribut; decls_seg: in atribut; decl: in atribut) is
begin
  decls:= new node(nd_decls);
  decls.decls_decls:= decls_seg;
  decls.decls_decl:= decl;
end rs_Decls;


procedure rs_Decl(decl: out atribut; decl_real: in atribut) is
begin
  decl:=new node(nd_decl);
  decl.decl_real:= decl_real;
end rs_Decl;


procedure rs_Decl_Var(decl: out atribut; lista_id: in atribut; tipus: in atribut) is
begin
  decl:= new node(nd_decl_var);
  decl.dvar_lid:= lista_id;
  decl.dvar_tipus:= tipus;
end rs_Decl_Var;


procedure rs_Decl_Const(decl: out atribut; lid_const: in atribut;
                        tipus: in atribut; valor: in atribut) is
begin
  decl:= new node(nd_decl_const);
  decl.dconst_lid:= lid_const;
  decl.dconst_tipus:= tipus;
  decl.dconst_valor:= valor;
end rs_Decl_Const;


procedure rs_Idx_neg(idx: out atribut; idx_cont: in atribut) is
begin
  idx:= new node(nd_idx);
  idx.idx_cont:= idx_cont;
  idx.idx_tipus:= negatiu;
end rs_Idx_neg;
```

```
procedure rs_Idx_pos(idx: out atribut; idx_cont: in atribut) is
begin
  idx:= new node(nd_idx);
  idx.idx_cont:= idx_cont;
  idx.idx_tipus:= positiu;
end rs_Idx_pos;

procedure rs_Idx_Cont(idx_cont: out atribut; valor: in atribut) is
begin
  idx_cont:= new node(nd_idx_cont);
  idx_cont.idxc_valor:= valor;
end rs_Idx_Cont;


procedure rs_Decl_T(decl: out atribut; id_type: in atribut; decl_cont: in atribut) is
begin
  decl:= new node(nd_decl_t);
  decl.dt_id:= id_type;
  decl.dt_cont:= decl_cont;
end rs_Decl_T;


procedure rs_Decl_T_Cont(decl: out atribut; info: in atribut) is
begin
  case info.tn is
    when nd_rang =>
      decl:= new node(nd_decl_t_cont_type);
      decl.dtcont_rang:= info;

    when nd_dcamps   =>
      decl:= new node(nd_decl_t_cont_record);
      decl.dtcont_camps:= info;

    when others =>
      null;
  end case;
end rs_Decl_T_Cont;


procedure rs_Decl_T_Cont(decl: out atribut; rang_array: in atribut;
                         tipus_array: in atribut) is
begin
  decl:= new node(nd_decl_t_cont_arry);
  decl.dtcont_idx:= rang_array;
  decl.dtcont_tipus:= tipus_array;
end rs_Decl_T_Cont;

procedure rs_DCamps(camps: out atribut; camp_seg: in atribut; camp: in atribut) is
begin
  camps:= new node(nd_dcamps);
  camps.dcamps_dcamps:= camp_seg;
  camps.dcamps_dcamp:= camp;
end rs_DCamps;


procedure rs_DCamps(camps: out atribut; camp: in atribut) is
begin
  camps:= new node(nd_dcamps);
  camps.dcamps_dcamps:= new node(nd_null);
  camps.dcamps_dcamp:= camp;
end rs_DCamps;


procedure rs_DCamp(camp: out atribut; var: in atribut) is
begin
  camp:= new node(nd_dcamp);
  camp.dcamp_decl:= var;
end rs_DCamp;


procedure rs_Rang(rang: out atribut; id_type: in atribut;
                  linf: in atribut; lsup: in atribut) is
begin
  rang:=new node(nd_rang);
  rang.rang_id:=id_type.id_id;
  rang.rang_linf:=linf;
  rang.rang_lsup:=lsup;
end rs_Rang;
```

```
-- Sentencies

procedure rs_Sents(sents: out atribut; sent: in atribut) is
begin
  sents:= new node(nd_sents);
  sents.sents_cont:= sent;
end rs_Sents;


procedure rs_Sent_Nob(sents: out atribut; sent_cont: in atribut; sent: in atribut) is
begin
  sents:= new node(nd_sents_nob);
  sents.snb_snb:= sent_cont;
  sents.snb_sent:= sent;
end rs_Sent_Nob;


procedure rs_Sent_Nob(sents: out atribut; sent: in atribut) is
begin
  sents:= new node(nd_sents_nob);
  sents.snb_snb:= new node(nd_null);
  sents.snb_sent:= sent;
end rs_Sent_Nob;


procedure rs_Sent(sent: out atribut; stipus: in atribut) is
begin
  sent:= new node(nd_sent);
  sent.sent_sent:= stipus;
end rs_Sent;


procedure rs_SIter(sent: out atribut; expr: in atribut; sents: in atribut) is
begin
  sent:= new node(nd_siter);
  sent.siter_expr:= expr;
  sent.siter_sents:= sents;
end rs_SIter;


procedure rs_SCond(sent: out atribut; expr: in atribut; sents: in atribut)
is
begin
  sent:= new node(nd_scond);
  sent.scond_expr:= expr;
  sent.scond_sents:= sents;
  sent.scond_esents:= new node(nd_null);
end rs_SCond;


procedure rs_SCond(sent: out atribut; expr: in atribut; sents_if: in atribut;
                   sents_else: in atribut) is
begin
  sent:= new node(nd_scond);
  sent.scond_expr:= expr;
  sent.scond_sents:= sents_if;
  sent.scond_esents:= sents_else;
end rs_SCond;


procedure rs_SCrida(sent: out atribut; ref: in atribut) is
begin
  sent:= new node(nd_scrida);
  sent.scrida_ref:= ref;
end rs_SCrida;


procedure rs_SAssign(sent: out atribut; ref: in atribut; expr: in atribut)
is
begin
  sent:= new node(nd_sassign);
  sent.sassign_ref:= ref;
  sent.sassign_expr:= expr;
end rs_SAssign;
```

```
-- Expressio

procedure rs_LExpr(lexpr: out atribut; cont: in atribut; expr: in atribut) is
begin
  lexpr:= new node(nd_lexpr);
  lexpr.lexpr_cont:= cont;
  lexpr.lexpr_expr:= expr;
end rs_LExpr;


procedure rs_LExpr(lexpr: out atribut; expr: in atribut) is
begin
  lexpr:= new node(nd_lexpr);
  lexpr.lexpr_cont:= new node(nd_null);
  lexpr.lexpr_expr:= expr;
end rs_LExpr;


procedure rs_Expr(expr: out atribut; cont: in atribut) is
begin
  expr:= new node(nd_expr);
  expr.expr_e:= cont;
end rs_Expr;


procedure rs_EAnd(expr: out atribut; ee: in atribut; ed: in atribut) is
begin
  expr:= new node(nd_and);
  expr.e_ope:= ee;
  expr.e_opd:= ed;
end rs_EAnd;


procedure rs_EOr(expr: out atribut; ee: in atribut; ed: in atribut) is
begin
  expr:= new node(nd_or);
  expr.e_ope:= ee;
  expr.e_opd:= ed;
end rs_EOr;


procedure rs_EOpo(expr: out atribut; ee: in atribut; ed: in atribut; op: in atribut) is
begin
  expr:= new node(nd_eop);
  expr.eop_ope:= ee;
  expr.eop_opd:= ed;
  expr.eop_operand:= op.orel_tipus;
end rs_EOpo;

procedure rs_EOps(expr: out atribut; ee: in atribut; ed: in atribut) is
begin
  expr:= new node(nd_eop);
  expr.eop_ope:= ee;
  expr.eop_opd:= ed;
  expr.eop_operand:= decls.d_arbre.sum;
end rs_EOps;

procedure rs_EOpr(expr: out atribut; ee: in atribut; ed: in atribut) is
begin
  expr:= new node(nd_eop);
  expr.eop_ope:= ee;
  expr.eop_opd:= ed;
  expr.eop_operand:= decls.d_arbre.res;
end rs_EOpr;

procedure rs_EOpp(expr: out atribut; ee: in atribut; ed: in atribut) is
begin
  expr:= new node(nd_eop);
  expr.eop_ope:= ee;
  expr.eop_opd:= ed;
  expr.eop_operand:= decls.d_arbre.prod;
end rs_EOpp;

procedure rs_EOpq(expr: out atribut; ee: in atribut; ed: in atribut) is
begin
  expr:= new node(nd_eop);
  expr.eop_ope:= ee;
  expr.eop_opd:= ed;
  expr.eop_operand:= decls.d_arbre.quoci;
end rs_EOpq;
```

```
  procedure rs_EOpm(expr: out atribut; ee: in atribut; ed: in atribut) is
  begin
    expr:= new node(nd_eop);
    expr.eop_ope:= ee;
    expr.eop_opd:= ed;
    expr.eop_operand:= decls.d_arbre.modul;
  end rs_EOpm;

  procedure rs_EOpnl(expr: out atribut; ed: in atribut) is
  begin
    expr:= new node(nd_eop);
    expr.eop_ope:= new node(nd_null);
    expr.eop_opd:= ed;
    expr.eop_operand:= decls.d_arbre.neg_log;
  end rs_EOpnl;

  procedure rs_EOpna(expr: out atribut; ed: in atribut) is
  begin
    expr:= new node(nd_eop);
    expr.eop_ope:= new node(nd_null);
    expr.eop_opd:= ed;
    expr.eop_operand:= decls.d_arbre.neg_alg;
  end rs_EOpna;

  procedure rs_EOp(expr: out atribut; ed: in atribut) is
  begin
    expr:= new node(nd_eop);
    expr.eop_ope:= new node(nd_null);
    expr.eop_opd:= ed;
    expr.eop_operand:= nul;
  end rs_EOp;

  procedure rs_ET(expr: out atribut; e: in atribut) is
  begin
    expr:= new node(nd_et);
    expr.et_cont:= e;
  end rs_ET;

  -- Referències
  procedure rs_Ref(ref: out atribut; ref_id: in atribut; qs: in atribut) is
  begin
    ref:= new node(nd_ref);
    ref.ref_id:= ref_id;
    ref.ref_qs:= qs;
  end rs_Ref;

  procedure rs_Qs(qs: out atribut; qs_in: in atribut; q: in atribut) is
  begin
    qs:= new node(nd_qs);
    qs.qs_qs:= qs_in;
    qs.qs_q:= q;
  end rs_Qs;

  procedure rs_Q(q: out atribut; contingut: in atribut) is
  begin
    q:= new node(nd_q);
    q.q_contingut:= contingut;
  end rs_Q;

  -- Llistes
  procedure rs_Lid(lid: out atribut; id_seg: in atribut; id: in atribut)
  is
  begin
    lid:= new node(nd_lid);
    lid.lid_seg:= id_seg;
    lid.lid_id:= id;
  end rs_Lid;


  procedure rs_Lid(lid: out atribut; id: in atribut) is
  begin
    lid:= new node(nd_lid);
    lid.lid_seg:= new node(nd_null);
    lid.lid_id:= id;
  end rs_Lid;
end semantica.c_arbre;
```

```
with decls.d_descripcio; use decls.d_descripcio;

package semantica.c_tipus is

  procedure comprovacio_tipus (err: out boolean);
  -- valors de true i false a la tv
  procedure posa_entorn_standard (c, f: out num_var);

end semantica.c_tipus;



--BODY
with Ada.Text_IO; use Ada.Text_IO;

with decls; use decls;
with decls.d_descripcio; use decls.d_descripcio;
with decls.d_tnoms; use decls.d_tnoms;
with decls.d_tsimbols; use decls.d_tsimbols;
with semantica.missatges; use semantica.missatges;


package body semantica.c_tipus is

  use Pila_Procediments;

  --DEFINICIONS
  procedure ct_decls(nd_decls: in out pnode);
  procedure ct_decl_tipus(decl_tipus: in out pnode);

  procedure ct_decl_var(decl_var: in out pnode);
  procedure ct_lid_var(lid_var: in out pnode; idt: in id_nom);

  procedure ct_decl_const(decl_const: in out pnode);
  procedure ct_lid_const(lid_const: in out pnode; desc: in decls.d_descripcio.descripcio);
  procedure ct_valor(nd_idx: in out pnode; id_tipus: out id_nom;
                     tsb: out decls.d_descripcio.tipus_subjacent;
                     v: out valor; pos: in out posicio);

  procedure ct_rang(decl_rang: in out pnode);

  procedure ct_record(nd_record: in out pnode);
  procedure ct_dcamps(nd_dcamps: in out pnode;idr: in id_nom;desp: in out despl);
  procedure ct_dcamp(nd_dcamp: in out pnode;idr: in id_nom;desp: in out despl);
  procedure ct_dcamp_lid(nd_lid: in out pnode; idt,idr: in id_nom;
                         desp: in out despl;ocup: in despl);

  procedure ct_array(nd_array: in out pnode);
  procedure ct_array_idx(nd_lidx: in out pnode;id_array: in id_nom; num_comp: in out valor;
                         b: in out valor);

  procedure ct_decl_proc(nd_procediment: in out pnode);
  procedure ct_decl_args(nd_args: in out pnode;id_proc: in id_nom; nargs: in out natural);
  procedure ct_decl_arg(nd_lid_arg: in out pnode;id_proc,id_tipus: in id_nom;
                        mode: in tmode; nargs: in out natural);

  --Sentencias
  procedure ct_sents(nd_sents: in out pnode);
  procedure ct_sents_nob(nd_sents_nob: in out pnode);
  procedure ct_sent_iter(nd_sent: in out pnode);
  procedure ct_sent_cond(nd_sent: in out pnode);
  procedure ct_sent_crida(nd_sent: in out pnode);
  procedure ct_sent_assign(nd_sent: in out pnode);

  --Referencias
  procedure ct_ref(nd_ref: in out pnode; id_base: out id_nom; id_tipus: out id_nom;
                   pos: in out posicio);
  procedure ct_qs(nd_qs: in out pnode; id_base: in id_nom; id_tipus: in out id_nom;
                  pos: in out posicio);
  procedure ct_q(nd_q: in out pnode; id_base: in id_nom; id_tipus: in out id_nom;
                 pos: in out posicio);
  procedure ct_qs_proc(nd_qs: in out pnode; id_base: in id_nom; pos: in out posicio);

  procedure ct_lexpr_proc(nd_lexpr: in out pnode; id_base: in id_nom;it: in out iterador_arg;
                          pos: in out posicio);
  procedure ct_lexpr_array(nd_lexpr: in out pnode; id_base: in id_nom; id_tipus: in out id_nom;
                           it: in out iterador_index; pos: in out posicio);
  procedure ct_expr(nd_expr: in out pnode; id_texpr: out id_nom; tsb_expr: out tipus_subjacent;
                    esvar: out boolean; pos: in out posicio);
```

```
--Expressions
procedure ct_e(nd_e: in out pnode; id_texpr: out id_nom; tsb_expr: out tipus_subjacent;
               esvar: out boolean; pos: in out posicio);

procedure ct_eop(nd_e: in out pnode; id_texpr: out id_nom; tsb_expr: out tipus_subjacent;
                 esvar: out boolean; pos: in out posicio);
procedure ct_eop_op_rel(nd_e: in out pnode; id_texpr: out id_nom;
                        tsb_expr: out tipus_subjacent; esvar: out boolean;
                        pos: in out posicio);
procedure ct_eop_arit(nd_e: in out pnode; id_texpr: out id_nom;
                      tsb_expr: out tipus_subjacent; esvar: out boolean;
                      pos: in out posicio; op: in operand );
procedure ct_eop_neg_log(nd_e: in out pnode; id_texpr: out id_nom;
                         tsb_expr: out tipus_subjacent; esvar: out boolean;
                         pos: in out posicio);
procedure ct_eop_neg_arit(nd_e: in out pnode; id_texpr: out id_nom;
                          tsb_expr: out tipus_subjacent; esvar: out boolean;
                          pos: in out posicio);

procedure ct_et(nd_e: in out pnode; id_texpr: out id_nom; tsb_expr: out tipus_subjacent;
                esvar: out  boolean; pos: in out posicio);
procedure ct_et_lit(nd_lit: in out pnode; id_texpr: out id_nom;
                    tsb_expr: out tipus_subjacent; esvar: out boolean; pos: in out posicio);
procedure ct_et_ref(nd_ref: in out pnode; id_texpr: out id_nom;
                    tsb_expr: out tipus_subjacent; esvar: out boolean; pos: in out posicio);

--Auxiliar
procedure tipus_compatible(id_tipus1, id_tipus2: in id_nom; tsb1, tsb2: in tipus_subjacent;
                           id_texp: out id_nom; tsb_exp: out tipus_subjacent;
                           error: out boolean);
```

```
ERROR: boolean:= false;


procedure comprovacio_tipus (err: out boolean) is
begin
  buida(pproc);
  ct_decl_proc(root.p);
  err:= ERROR;
end comprovacio_tipus;


--S'inicialitzen els tipus basics, altres valors predefinits(true,false)
--i els procediments d'entrada/sortida
procedure posa_entorn_standard (c,f: out num_var) is
  idb,idtr,idf, id_arg:id_nom;
  idint,idchar: id_nom;
  idstdio: id_nom;
  desc, desc_arg: descripcio;
  error: boolean;
  prof: profunditat;
  t: num_var;
  p: num_proc;
begin
  prof:= get_prof(ts);

  --Booleanes
  put(tn, "boolean", idb);
  desc:= (td=>dtipus, dt=>(tsb=>tsb_bool, ocup=>4, linf=>-1, lsup=>0));
  put(ts, idb, desc, error);
  if error then
    ERROR:= true;
    missatges_ct_error_intern((fila=>18, columna=>5), "posa_entorn_standart");
  end if;

  put(tn, "true", idtr);
  desc:= (td=>dconst, tc=>idb, vc=>-1);
  put(ts, idtr, desc, error);
  if error then
    ERROR:= true;
    missatges_ct_error_intern((fila=>26, columna=>5), "posa_entorn_standart");
  end if;
  nova_var_const(nv, tv, -1, tsb_ent, c);

  put(tn, "false", idf);
  desc:= (td=>dconst, tc=>idb, vc=>0);
  put(ts, idf, desc, error);
  if error then
    ERROR:= true;
    missatges_ct_error_intern((fila=>35, columna=>5), "posa_entorn_standart");
  end if;
  nova_var_const(nv, tv, 0, tsb_ent, f);

  --Enters
  put(tn, "integer", idint);
  desc:= (td=>dtipus, dt=>(tsb=>tsb_ent, ocup=>4, linf=>valor'First, lsup=>valor'Last));
  put(ts, idint, desc, error);

  if error then
    ERROR:= true;
    missatges_ct_error_intern((fila=>46, columna=>5), "posa_entorn_standart");
  end if;

  --Caracters
  put(tn, "character", idchar);
  desc:= (td=>dtipus, dt=>(tsb=>tsb_car, ocup=>4, linf=>Character'pos(Character'First),
          lsup=>Character'pos(Character'Last)));
  put(ts, idchar, desc, error);
  if error then
    ERROR:= true;
    missatges_ct_error_intern((fila=>55, columna=>5), "posa_entorn_standart");
  end if;
```

```
--STDIO
--putc
put(tn, "putc", idstdio);
nou_proc_std(np, tp, idstdio, prof, 1, p);
desc:= (td=>dproc, np=> p);
empila(pproc, np);
put(ts, idstdio, desc, error);
if error then
  ERROR:= true;
  missatges_ct_error_intern((fila=>68, columna=>5), "posa_entorn_standart");
end if;

put(tn, "c", id_arg);
nova_var(nv, tv, tp, np, ocup_char, t);
desc_arg:=(td=>dargc, ta=> idchar, na=> t); -- character::{ocup: 4Bytes, despl: 0}
put_arg(ts, idstdio, id_arg, desc_arg, error);

--puti
put(tn, "puti", idstdio);
nou_proc_std(np, tp, idstdio, prof, 1, p);
desc:= (td=>dproc, np=> p);
empila(pproc, np);
put(ts, idstdio, desc, error);
if error then
  ERROR:= true;
  missatges_ct_error_intern((fila=>85, columna=>5), "posa_entorn_standart");
end if;

put(tn, "n", id_arg);
nova_var(nv, tv, tp, np, ocup_ent, t);
desc_arg:=(td=>dargc, ta=> idint, na=> t); -- integer::{ocup: 4Bytes, despl: 0}
put_arg(ts, idstdio, id_arg, desc_arg, error);

--puts
put(tn, "puts", idstdio);
nou_proc_std(np, tp, idstdio, prof, 1, p);
desc:= (td=>dproc, np=> p);
empila(pproc, np);
put(ts, idstdio, desc, error);
if error then
  ERROR:= true;
  missatges_ct_error_intern((fila=>103, columna=>5), "posa_entorn_standart");
end if;

put(tn, "s", id_arg);
nova_var(nv, tv, tp, np, ocup_ent, t); -- passam l'ids
desc_arg:=(td=>dargc, ta=> null_id, na=> t); -- string::{ocup: 4B, despl: 0}
put_arg(ts, idstdio, id_arg, desc_arg, error);

--newline
put(tn, "new_line", idstdio);
nou_proc_std(np, tp, idstdio, prof, 0, p);
desc:= (td=>dproc, np=> p);
empila(pproc, np);
put(ts, idstdio, desc, error);
if error then
  ERROR:= true;
  missatges_ct_error_intern((fila=>120, columna=>5), "posa_entorn_standart");
end if;

--geti
put(tn, "geti", idstdio);
nou_proc_std(np, tp, idstdio, prof, 0, p);
desc:= (td=>dproc, np=> p); -- ocupacio? 0 params?
put(ts, idstdio, desc, error);
if error then
  ERROR:= true;
  missatges_ct_error_intern((fila=>130, columna=>5), "posa_entorn_standart");
end if;

put(tn, "n", id_arg);
nova_var(nv, tv, tp, np, ocup_ent, t);
desc_arg:=(td=>dvar, tv=> idint, nv=> t);
put_arg(ts, idstdio, id_arg, desc_arg, error);
```

```
  --getc -> per caracters de 4 bytes(com es el nostre cas)
  put(tn, "getc", idstdio);
  nou_proc_std(np, tp, idstdio, prof, 0, p);
  desc:= (td=>dproc, np=> p);
  empila(pproc, np);
  put(ts, idstdio, desc, error);
  if error then
    ERROR:= true;
    missatges_ct_error_intern((fila=>146, columna=>5), "posa_entorn_standart");
  end if;

  put(tn, "c", id_arg);
  nova_var(nv, tv, tp, np, ocup_char, t);
  desc_arg:=(td=>dvar, tv=> idchar, nv=> t);
  put_arg(ts, idstdio, id_arg, desc_arg, error);

  --getcc -> per caracters de 1 byte( en cas que lo necessitem)
  put(tn, "getcc", idstdio);
  nou_proc_std(np, tp, idstdio, prof, 0, p);
  desc:= (td=>dproc, np=> p);
  empila(pproc, np);
  put(ts, idstdio, desc, error);
  if error then
    ERROR:= true;
    missatges_ct_error_intern((fila=>162, columna=>5), "posa_entorn_standart");
  end if;

  put(tn, "c", id_arg);
  nova_var(nv, tv, tp, np, ocup_char, t);
  desc_arg:=(td=>dvar, tv=> idchar, nv=> t);
  put_arg(ts, idstdio, id_arg, desc_arg, error);
end posa_entorn_standard;

--Declaracions
procedure ct_decls(nd_decls: in out pnode) is
  nd_decl: pnode;
begin
  if nd_decls.decls_decls.tn /= nd_null then
    ct_decls(nd_decls.decls_decls);
  end if;
  nd_decl:= nd_decls.decls_decl.decl_real;
  case nd_decl.tn is
    when nd_proc=>
      ct_decl_proc(nd_decl);

    when nd_decl_var=>
      ct_decl_var(nd_decl);

    when nd_decl_t=>
      ct_decl_tipus(nd_decl);

    when nd_decl_const=>
      ct_decl_const(nd_decl);

    when others =>
      ERROR:= true;
      missatges_ct_error_intern((fila=>22, columna=>5), "ct_decls");
      return;
  end case;
end ct_decls;


procedure ct_decl_tipus(decl_tipus: in out pnode) is
begin
  case decl_tipus.dt_cont.tn is
    when nd_decl_t_cont_type=>
      ct_rang(decl_tipus);

    when nd_decl_t_cont_record=>
      ct_record(decl_tipus);

    when nd_decl_t_cont_arry=>
      ct_array(decl_tipus);

    when others =>
      ERROR:= true;
      missatges_ct_error_intern((fila=>15, columna=>5), "ct_decl_tipus");
      return;
  end case;
end ct_decl_tipus;
```

```
procedure ct_decl_var(decl_var: in out pnode) is
  id_tipus: id_nom;
  d_tipus: descripcio;
begin
  id_tipus:= decl_var.dvar_tipus.id_id;
  d_tipus:= get(ts, id_tipus);
  if d_tipus.td /= dtipus then
    ERROR:= true;
    missatges_desc_no_es_tipus(decl_var.dvar_tipus.id_pos, id_tipus);
    return;
  end if;
  ct_lid_var(decl_var.dvar_lid, id_tipus);
end ct_decl_var;


procedure ct_lid_var(lid_var: in out pnode; idt: in id_nom) is
  id_var: id_nom;
  d_var,desc: descripcio;
  error: boolean;
  t: num_var;
begin
  if lid_var.lid_seg.tn /= nd_null then
    ct_lid_var(lid_var.lid_seg, idt);
  end if;
  id_var:= lid_var.lid_id.id_id; --L'identificador de la var
  desc:= get(ts, idt);
  nova_var(nv, tv, tp, cim(pproc), desc.dt.ocup, t);
  d_var:= (td=>dvar, tv=>idt, nv=> t);
  put(ts, id_var, d_var, error);
  if error then
    ERROR:= true;
    missatges_conflictes_declaracio(lid_var.lid_id.id_pos, id_var);
  end if;
end ct_lid_var;


procedure ct_decl_const(decl_const: in out pnode) is
  id_tipus: id_nom;
  d_tipus, desc: descripcio;
  id_valor: id_nom;
  v_valor: valor;
  tsb_valor: tipus_subjacent;
  pos_valor: posicio:= (0, 0);
begin
  id_tipus:= decl_const.dconst_tipus.id_id;
  d_tipus:= get(ts, id_tipus);

  if d_tipus.td /= dtipus then
    ERROR:= true;
    missatges_desc_no_es_tipus(decl_const.dconst_tipus.id_pos, id_tipus);
  end if;

  if d_tipus.dt.tsb > tsb_ent then
    ERROR:= true;
    missatges_operacio_amb_escalar(decl_const.dconst_tipus.id_pos);
    return;
  end if;

  ct_valor(decl_const.dconst_valor, id_valor, tsb_valor, v_valor, pos_valor);
  if ERROR then return; end if;
  if id_valor = null_id and then tsb_valor /= d_tipus.dt.tsb then
    ERROR:= true;
    missatges_tipus_incosistent_lit(pos_valor, id_tipus, tsb_valor);
    return;
  end if;

  if id_valor /= null_id and then id_valor /= id_tipus then
    ERROR:= true;
    missatges_tipus_incosistent_id(pos_valor, id_tipus, id_valor);
    return;
  end if;

  if v_valor < d_tipus.dt.linf or v_valor > d_tipus.dt.lsup then
    ERROR:= true;
    missatges_valor_fora_rang(pos_valor, id_tipus);
  end if;

  desc:= (td=>dconst, tc=>id_tipus, vc=>v_valor);
  ct_lid_const(lid_const=>decl_const.dconst_lid, desc=> desc);
end ct_decl_const;
```

```
procedure ct_lid_const(lid_const: in out pnode; desc: in descripcio) is
  id_const: id_nom;
  error: boolean;
begin
  if lid_const.lid_seg.tn /= nd_null then
    ct_lid_const(lid_const.lid_seg, desc);
  end if;
  id_const:= lid_const.lid_id.id_id;
  put(ts, id_const, desc, error);
  if error then
    ERROR:= true;
    missatges_conflictes_declaracio(lid_const.lid_id.id_pos, id_const);
  end if;
end ct_lid_const;


procedure ct_rang(decl_rang: in out pnode) is
  id_tipus, id_rang: id_nom;
  desc_tipus: descripcio;

  id_valor1, id_valor2: id_nom;
  v_valor1, v_valor2: valor;
  tsb_valor1, tsb_valor2: tipus_subjacent;
  pos_valor1, pos_valor2: posicio:= (0, 0);

  desc:descripcio;
  nd_rang: pnode;
  error: boolean;
begin
  nd_rang:= decl_rang.dt_cont.dtcont_rang;
  id_rang:= decl_rang.dt_id.id_id;
  id_tipus:= nd_rang.rang_id;
  desc_tipus:= get(ts, id_tipus);

  if desc_tipus.td /= dtipus then
    ERROR:= true;
    missatges_desc_no_es_tipus(decl_rang.dt_id.id_pos, id_tipus);
  end if;

  if desc_tipus.dt.tsb > tsb_ent then
    ERROR:= true;
    missatges_operacio_amb_escalar(nd_rang.rang_pos);
    return;
  end if;

  ct_valor(nd_rang.rang_linf, id_valor1, tsb_valor1, v_valor1, pos_valor1);
  if ERROR then return; end if;
  if id_valor1 = null_id and then tsb_valor1 /= desc_tipus.dt.tsb then
    ERROR:= true;
    missatges_tipus_incosistent_lit(pos_valor1, id_tipus, tsb_valor1);
  end if;

  if id_valor1 /= null_id and then id_valor1 /= id_tipus then
    ERROR:= true;
    missatges_tipus_incosistent_id(pos_valor1, id_tipus, id_valor1);
    return;
  end if;

  ct_valor(nd_rang.rang_lsup, id_valor2, tsb_valor2, v_valor2, pos_valor2);
  if id_valor2 = null_id and then tsb_valor2 /= desc_tipus.dt.tsb then
    ERROR:= true;
    missatges_tipus_incosistent_lit(pos_valor2, id_tipus, tsb_valor2);
    return;
  end if;

  if id_valor2 /= null_id and then id_valor2 /= id_tipus then
    ERROR:= true;
    missatges_tipus_incosistent_id(pos_valor2, id_tipus, id_valor2);
  end if;

  if v_valor1 > v_valor2 then
    ERROR:= true;
    missatges_rang_incorrecte(pos_valor1);
  end if;

  if v_valor1 < desc_tipus.dt.linf then
    ERROR:= true;
    missatges_valor_fora_rang(pos_valor1, id_tipus);
  end if;
```

```
  if v_valor2 > desc_tipus.dt.lsup then
    ERROR:= true;
    missatges_valor_fora_rang(pos_valor2, id_tipus);
  end if;

  case desc_tipus.dt.tsb is

    when tsb_ent =>
      desc:= (td=>dtipus,
      dt=>(tsb=> tsb_ent, ocup=> desc_tipus.dt.ocup, linf=>v_valor1, lsup=>v_valor2));

    when tsb_car =>
      desc:= (td=>dtipus,
      dt=>(tsb=> tsb_car, ocup=> desc_tipus.dt.ocup, linf=>v_valor1, lsup=>v_valor2));

    when tsb_bool =>
      desc:= (td=>dtipus,
      dt=>(tsb=> tsb_bool, ocup=> desc_tipus.dt.ocup, linf=>v_valor1, lsup=>v_valor2));

    when others =>
      ERROR:= true;
      missatges_ct_error_intern((fila=>85, columna=>5), "ct_rang");
  end case;

  put(ts, id_rang, desc, error);
  if error then
    ERROR:= true;
    missatges_conflictes_declaracio(decl_rang.dt_id.id_pos, id_rang);
  end if;
end ct_rang;

procedure ct_valor(nd_idx: in out pnode; id_tipus: out id_nom; tsb: out tipus_subjacent;
                   v: out valor; pos: in out posicio) is
  desc, d_tipus: descripcio;
  p: pnode;
begin
  p:= nd_idx.idx_cont.idxc_valor;
  case p.tn is
    when nd_id =>
      desc:= get(ts, p.id_id);
      if desc.td /= dconst then
        ERROR:= true;
        missatges_assignacio_incorrecta(p.id_pos);
        return;
      end if;
      d_tipus:= get(ts, desc.tc);

      if nd_idx.idx_tipus = negatiu then
        if d_tipus.dt.tsb /= tsb_ent then
          ERROR:= true;
          missatges_operador_tipus(p.id_pos, d_tipus.dt.tsb, neg_alg);
        end if;
        v:= -desc.vc;
      else
        v:= desc.vc;
      end if;

      id_tipus:= desc.tc;
      tsb:= d_tipus.dt.tsb;
      pos:= p.id_pos;

    when nd_lit =>
      id_tipus:= null_id;
      tsb:= p.lit_tipus;
      if nd_idx.idx_tipus = negatiu then
        if tsb /= tsb_ent then
          ERROR:= true;
          missatges_operador_tipus(p.lit_pos, tsb, neg_alg);
        end if;
        v:= -p.lit_val;
      else
        v:= p.lit_val;
      end if;
      pos:= p.lit_pos;

    when others =>
      ERROR:= true;
      missatges_ct_error_intern((fila=>46, columna=>5), "ct_valor");
  end case;
end ct_valor;
```

```
procedure ct_record(nd_record: in out pnode) is
  desc_record: descripcio;
  id_record: id_nom;
  error: boolean;
  desp: despl:= 0; --El desplacament dels camps
begin
  id_record:= nd_record.dt_id.id_id;
  desc_record:= (td=>dtipus, dt=>(tsb=> tsb_rec, ocup=> 0));
  put(ts, id_record, desc_record, error);
  if error then
    ERROR:= true;
    missatges_conflictes_declaracio(nd_record.dt_id.id_pos, id_record);
    return;
  end if;

  ct_dcamps(nd_record.dt_cont.dtcont_camps, id_record, desp);
  --Actualitzacio de l'ocupacio del record
  desc_record:= (td=>dtipus, dt=>(tsb=> tsb_rec, ocup=> desp));
  update(ts, id_record, desc_record);
end ct_record;


procedure ct_dcamps(nd_dcamps: in out pnode;idr: in id_nom;desp: in out despl) is
begin
  if nd_dcamps.dcamps_dcamps.tn /= nd_null then
    ct_dcamps(nd_dcamps.dcamps_dcamps, idr, desp);
  end if;
  ct_dcamp(nd_dcamps.dcamps_dcamp, idr, desp);
end ct_dcamps;


procedure ct_dcamp(nd_dcamp: in out pnode;idr: in id_nom;desp: in out despl) is
  id_tipus: id_nom;
  desc_tipus: descripcio;
begin
  id_tipus:= nd_dcamp.dcamp_decl.dvar_tipus.id_id;
  desc_tipus:= get(ts, id_tipus);

  if desc_tipus.td /= dtipus then
    ERROR:= true;
    missatges_desc_no_es_tipus(nd_dcamp.dcamp_decl.dvar_tipus.id_pos, id_tipus);
    return;
  end if;

  ct_dcamp_lid(nd_dcamp.dcamp_decl.dvar_lid, id_tipus, idr, desp, desc_tipus.dt.ocup);
end ct_dcamp;


procedure ct_dcamp_lid(nd_lid: in out pnode; idt, idr: in id_nom; desp: in out despl;
                       ocup: in despl) is
  id_camp: id_nom;
  desc_camp: descripcio;
  error: boolean;
begin
  if nd_lid.lid_seg.tn /= nd_null then
    ct_dcamp_lid(nd_lid.lid_seg, idt, idr, desp, ocup);
  end if;
  id_camp:= nd_lid.lid_id.id_id;
  desc_camp:= (td=>dcamp, tcmp=>idt, dcmp=>desp);
  put_camp(ts, idr, id_camp, desc_camp, error);
  if error then
    ERROR:= true;
    missatges_conflictes_declaracio(nd_lid.lid_id.id_pos, id_camp);
  end if;
  desp:= desp+ocup;
end ct_dcamp_lid;
```

```
--ARRAYS
procedure ct_array(nd_array: in out pnode) is
  id_array: id_nom;
  desc_array: descripcio;
  id_tipus: id_nom;
  desc_tipus:descripcio;
  num_components: valor:= 0;
  ocup: despl;
  error: boolean;
  ib: valor:= 0;
begin
  id_array:= nd_array.dt_id.id_id;
  id_tipus:= nd_array.dt_cont.dtcont_tipus.id_id;
  desc_array:= (td=>dtipus, dt=> (tsb=>tsb_arr, ocup=>0, tcomp=>id_tipus, b=>0));
  put(ts, id_array, desc_array, error);

  if error then
    ERROR:= true;
    missatges_conflictes_declaracio(nd_array.dt_id.id_pos, id_array);
    return;
  end if;

  ct_array_idx(nd_array.dt_cont.dtcont_idx, id_array, num_components, ib);

  desc_tipus:= get(ts, id_tipus);
  if desc_tipus.td /= dtipus then
    ERROR:= true;
    missatges_desc_no_es_tipus(nd_array.dt_cont.dtcont_tipus.id_pos, id_tipus);
    return;
  end if;

  ocup:= despl(num_components)*desc_tipus.dt.ocup;
  desc_array:= (td=>dtipus, dt=> (tsb=>tsb_arr, ocup=>ocup, tcomp=>id_tipus, b=>ib));
  update(ts, id_array, desc_array);
end ct_array;


procedure ct_array_idx(nd_lidx: in out pnode;id_array: in id_nom;
                       num_comp: in out valor; b: in out valor) is
  desc_rang, desc_idx:descripcio;
  id_rang: id_nom;
  lnc,linf: valor;
begin
  if nd_lidx.lid_seg.tn /= nd_null then
    ct_array_idx(nd_lidx.lid_seg, id_array, num_comp, b);
  end if;

  id_rang:= nd_lidx.lid_id.id_id;
  desc_rang:= get(ts, id_rang);

  if desc_rang.td /= dtipus then
    ERROR:= true;
    missatges_desc_no_es_tipus(nd_lidx.lid_id.id_pos, id_rang);
    return;
  end if;

  if desc_rang.dt.tsb > tsb_ent then
    ERROR:= true;
    missatges_operacio_amb_escalar(nd_lidx.lid_id.id_pos);
    return;
  end if;

  linf:= desc_rang.dt.linf;
  lnc:= desc_rang.dt.lsup - linf + 1;
  b:= b * lnc + linf;

  desc_idx:= (td=>dindx, tind=>id_rang);
  put_index(ts, id_array, desc_idx);
  if num_comp = 0 then
    num_comp:= lnc;
  else
    num_comp:= num_comp * (lnc);
  end if;
end ct_array_idx;
```

```
procedure ct_decl_proc(nd_procediment: in out pnode) is
  id_proc, id_arg: id_nom;
  desc_proc,desc_arg: descripcio;
  error:boolean;
  it: iterador_arg;
  nargs: natural:= 0;
  despl_args: despl;
  t: num_var;
  p: num_proc;
  e: num_etiq;
begin
  id_proc:= nd_procediment.proc_cproc.cproc_id.id_id;
  nova_etiq(ne, e);
  -- 0 params i 0 ocupacio temporals
  nou_proc(np, tp, e, get_prof(ts), 0, p);
  desc_proc:= (td=>dproc, np=> p);
  put(ts, id_proc, desc_proc, error);
  if error then
    ERROR:= true;
    missatges_conflictes_declaracio(nd_procediment.proc_cproc.cproc_id.id_pos, id_proc);
    return;
  end if;

  if nd_procediment.proc_cproc.cproc_args.tn /= nd_null then
    ct_decl_args(nd_procediment.proc_cproc.cproc_args, id_proc, nargs);
  end if;

  nd_procediment.proc_cproc.cproc_np:= desc_proc.np;
  empila(pproc, desc_proc.np);

  enter_block(ts);
  -- el desplaçament del primer argument es bp + antic Disp + @RTN
  -- + ocup_ent * nargs (això es degut a que l'operació pushl no
  -- funciona exactament com un podria esperar)
  despl_args:= 3 * ocup_ent + ocup_ent * despl(nargs);
  first(ts, id_proc, it);
  while is_valid(it) loop
    -- ho feim aixi perque el càlcul anterior, deixa el punter a 4 posicions mes enllà
    -- d'on caldria
    despl_args:= despl_args - ocup_ent;
    nou_arg(nv, tv, tp, cim(pproc), despl_args, t);
    get(ts, it, id_arg, desc_arg);
    if desc_arg.td = dvar then
      desc_arg.nv:= t;
    else
      desc_arg.na:= t;
    end if;
    put(ts, id_arg, desc_arg, error);
    next(ts, it);
  end loop;
  -- tot i que aixo pot anar abans del desempila(pproc) aixi sembla mes adecuat
  act_proc_args(tp, cim(pproc), nargs);
  if nd_procediment.proc_decls.tn /= nd_null then
    ct_decls(nd_procediment.proc_decls);
  end if;
  if nd_procediment.proc_sents.tn /= nd_null then
    ct_sents(nd_procediment.proc_sents);
  end if;
  exit_block(ts);
  desempila(pproc);
end ct_decl_proc;


procedure ct_decl_args(nd_args: in out pnode;id_proc: in id_nom; nargs: in out natural) is
  id_tipus: id_nom;
  desc_tipus: descripcio;
begin
  if nd_args.args_args.tn /= nd_null then
    ct_decl_args(nd_args.args_args, id_proc, nargs);
  end if;

  id_tipus:= nd_args.args_arg.arg_tipus.id_id;
  desc_tipus:= get(ts, id_tipus);
  if desc_tipus.td /= dtipus then
    ERROR:= true;
    missatges_desc_no_es_tipus(nd_args.args_arg.arg_tipus.id_pos, id_tipus);
  end if;

  ct_decl_arg(nd_args.args_arg.arg_lid, id_proc, id_tipus, nd_args.args_arg.arg_mode, nargs);
end ct_decl_args;
```

```
procedure ct_decl_arg(nd_lid_arg: in out pnode; id_proc, id_tipus: in id_nom;
                      mode: in tmode; nargs: in out natural) is
  id_arg: id_nom;
  desc_arg,desc: descripcio;
  error: boolean;
begin
  if nd_lid_arg.lid_seg.tn /= nd_null then
    ct_decl_arg(nd_lid_arg.lid_seg, id_proc, id_tipus, mode, nargs);
  end if;
  nargs:= nargs + 1;
  desc:= get(ts, id_tipus);
  case mode is
    when md_in_out => desc_arg:= (td=>dvar, tv=>id_tipus, nv=>null_nv);
    when md_in => desc_arg:= (td=>dargc, ta=>id_tipus, na=> null_nv);
  end case;
  id_arg:= nd_lid_arg.lid_id.id_id;
  put_arg(ts, id_proc, id_arg, desc_arg, error);

  if error then
    ERROR:= true;
    missatges_conflictes_declaracio(nd_lid_arg.lid_id.id_pos, id_arg);
  end if;
end ct_decl_arg;


--Sentencias

procedure ct_sents(nd_sents: in out pnode) is
begin
  if nd_sents.sents_cont.tn = nd_sents_nob then
    ct_sents_nob(nd_sents.sents_cont);
  elsif nd_sents.sents_cont.tn /= nd_null then
    ERROR:= true;
    missatges_sent_buida;
  end if;
end ct_sents;

procedure ct_sents_nob(nd_sents_nob: in out pnode) is
begin
  if nd_sents_nob.snb_snb.tn /= nd_null then
    ct_sents_nob(nd_sents_nob.snb_snb);
  end if;
  case nd_sents_nob.snb_sent.sent_sent.tn is
    when nd_siter=>
      ct_sent_iter(nd_sents_nob.snb_sent.sent_sent);
    when nd_scond=>
      ct_sent_cond(nd_sents_nob.snb_sent.sent_sent);
    when nd_scrida=>
      ct_sent_crida(nd_sents_nob.snb_sent.sent_sent);
    when nd_sassign=>
      ct_sent_assign(nd_sents_nob.snb_sent.sent_sent);
    when others =>
      ERROR:= true;
      missatges_ct_error_intern((fila=>19, columna=>16), "ct_sents_nob");
  end case;
end ct_sents_nob;


procedure ct_sent_iter(nd_sent: in out pnode) is
  id_texpr: id_nom;
  tsb_expr: tipus_subjacent;
  expr_esvar: boolean;
  pos_exp: posicio:= (0, 0);
begin
  ct_expr(nd_sent.siter_expr, id_texpr, tsb_expr, expr_esvar, pos_exp);
  if tsb_expr /= tsb_bool then
    ERROR:= true;
    missatges_cond_bool(pos_exp, tsb_expr);
  end if;

  if nd_sent.siter_sents.tn /= nd_sents and nd_sent.siter_sents.tn /= nd_null then
    ERROR:= true;
    missatges_sent_buida;
  end if;

  if nd_sent.siter_sents.tn = nd_sents then
    ct_sents(nd_sent.siter_sents);
  end if;

end ct_sent_iter;
```

```
procedure ct_sent_cond(nd_sent: in out pnode) is
  id_texpr: id_nom;
  tsb_expr: tipus_subjacent;
  expr_esvar: boolean;
  pos_exp: posicio:= (0, 0);
begin
  ct_expr(nd_sent.scond_expr, id_texpr, tsb_expr, expr_esvar, pos_exp);
  if tsb_expr /= tsb_bool then
    ERROR:= true;
    missatges_cond_bool(pos_exp, tsb_expr);
  end if;

  if nd_sent.scond_sents.tn /= nd_sents and nd_sent.scond_sents.tn /= nd_null then
    ERROR::= true;
    missatges_sent_buida;
  end if;
  if nd_sent.scond_sents.tn = nd_sents then
    ct_sents(nd_sent.scond_sents);
  end if;

  if nd_sent.scond_esents.tn /= nd_null then
    if nd_sent.scond_esents.tn /= nd_sents and nd_sent.scond_esents.tn /= nd_null then
      ERROR:= true;
      missatges_sent_buida;
    end if;
  end if;
  if nd_sent.scond_esents.tn = nd_sents then
    ct_sents(nd_sent.scond_esents);
  end if;
end ct_sent_cond;


procedure ct_sent_crida(nd_sent: in out pnode) is
  id_base, id_tipus: id_nom;
  desc_ref: descripcio;
  pos_ref: posicio:= (0, 0);
begin
  ct_ref(nd_sent.scrida_ref, id_base, id_tipus, pos_ref);
  if ERROR then return; end if;
  desc_ref:= get(ts, id_base);
  if desc_ref.td /= dproc then
    missatges_no_proc(pos_ref);
    ERROR:=true;
    return;
  end if;
end ct_sent_crida;


procedure ct_sent_assign(nd_sent: in out pnode) is
  id_ref, id_tipus_ref, id_texpr: id_nom;
  desc_ref, desc_tipus_ref: descripcio;
  tsb_expr: tipus_subjacent;
  expr_esvar: boolean;
  pos_ref, pos_exp: posicio:= (0, 0);
begin
  ct_ref(nd_sent.sassign_ref, id_ref, id_tipus_ref, pos_ref);
  desc_ref:= get(ts, id_ref);
  if desc_ref.td /= dvar then
    ERROR:= true;
    missatges_assignacio_incorrecta(pos_ref);
    return;
  end if;
  ct_expr(nd_sent.sassign_expr, id_texpr, tsb_expr, expr_esvar, pos_exp);
  if id_texpr /= null_id then
    if id_tipus_ref /= id_texpr then
      ERROR:= true;
      missatges_tipus_incosistent_id(pos_exp, id_tipus_ref, id_texpr);
    end if;
  else
    desc_tipus_ref:= get(ts, id_tipus_ref);
    if desc_tipus_ref.dt.tsb /= tsb_expr then
      ERROR:= true;
      missatges_tipus_incosistent_lit(pos_exp, id_tipus_ref, tsb_expr);
    end if;
  end if;
  if tsb_expr > tsb_ent then
    ERROR:= true;
    missatges_operacio_amb_escalar(pos_exp);
  end if;
end ct_sent_assign;
```

```
procedure ct_ref(nd_ref: in out pnode; id_base: out id_nom; id_tipus: out id_nom;
                 pos: in out posicio) is
  it: iterador_arg;
  desc_ref, desc_ref_aux: descripcio;
  p: pnode;
begin
  id_base:= nd_ref.ref_id.id_id;
  pos:= nd_ref.ref_id.id_pos;
  desc_ref:= get(ts, id_base);
  case desc_ref.td is
    when dargc=>
      if nd_ref.ref_qs.tn /= nd_null then
        ct_qs(nd_ref.ref_qs, id_base, desc_ref.ta, pos);
      end if;
      id_tipus:= desc_ref.ta;
      desc_ref_aux:= desc_ref;

      desc_ref:= get(ts, desc_ref.ta);
      if desc_ref.td /= dtipus then
        ERROR:= true;
        missatges_ct_error_intern((fila=>19, columna=>20), "ct_ref::una cosa que ha pasat "
                                                & "el test com a tipus, ara ja no ho es");
        return;
      end if;

      if desc_ref.dt.tsb > tsb_ent then
        ERROR:= true;
        missatges_ct_error_intern((fila=>25, columna=>20), "ct_ref::una cosa que ha pasat "
                                                & "el test de constant, ara ja no ho es");
        return;
      end if;

      if not ERROR then
        p:= new node(nd_var);
        p.var_nv:= desc_ref_aux.na;
        p.var_ocup:= desc_ref.dt.ocup;
      end if;

    when dvar=>
      if nd_ref.ref_qs.tn /= nd_null then
        ct_qs(nd_ref.ref_qs, id_base, desc_ref.tv, pos);
      end if;
      id_tipus:= desc_ref.tv;

      if not ERROR then
        p:= new node(nd_var);
        p.var_nv:= desc_ref.nv;
        desc_ref_aux:= get(ts, desc_ref.tv);
        p.var_ocup:= desc_ref_aux.dt.ocup;
      end if;

    when dconst=>
      if nd_ref.ref_qs.tn /= nd_null then
        ct_qs(nd_ref.ref_qs, id_base, desc_ref.tc, pos);
      end if;
      id_tipus:= desc_ref.tc;
      desc_ref_aux:= desc_ref;

      desc_ref:= get(ts, desc_ref.tc);
      if desc_ref.td /= dtipus then
        ERROR:= true;
        missatges_ct_error_intern((fila=>59, columna=>20), "ct_ref::una cosa que ha pasat "
                                                & "el test com a tipus, ara ja no ho es");
        return;
      end if;

      if desc_ref.dt.tsb > tsb_ent then
        ERROR:= true;
        missatges_ct_error_intern((fila=>65, columna=>20), "ct_ref::una cosa que ha pasat "
                                                & "el test de constant, ara ja no ho es");
        return;
      end if;

      -- Si discriminam constants i variables a la TV no caldra discriminar entre nd_var
      -- i nd_const. Com que ho farem a la proxima etapa, ja no ens escarrassam massa
      if not ERROR then
        p:= new node(nd_var);
        nova_var_const(nv, tv, desc_ref_aux.vc, desc_ref.dt.tsb, p.var_nv);
        p.var_ocup:= desc_ref.dt.ocup;
      end if;
```

```
    when dproc =>
      --Comprovam si el procediment te arguments
      first(ts, id_base, it);
      if is_valid(it) then
        if nd_ref.ref_qs.tn = nd_null then
          ERROR:= true;
          missatges_menys_arguments_proc(pos, id_base);
          return;
        end if;
        ct_qs_proc(nd_ref.ref_qs, id_base, pos);
      else
        if nd_ref.ref_qs.tn /= nd_null then
          ERROR:= true;
          missatges_massa_arguments_proc(pos, id_base);
          return;
        end if;
      end if;
      id_tipus:= null_id;

      if not ERROR then
        p:= new node(nd_iproc);
        p.iproc_np:= desc_ref.np;
      end if;

    when others=>
      ERROR:= true;
      missatges_no_definida(pos, id_base);
  end case;

  if not ERROR then
    nd_ref.ref_id:= p;
  end if;

end ct_ref;


procedure ct_qs_proc(nd_qs: in out pnode; id_base: in id_nom; pos: in out posicio) is
  it: iterador_arg;
begin
  if nd_qs.qs_qs.tn /= nd_null then
    ERROR:= true;
    missatges_proc_mult_parentesis(pos);
  end if;

  first(ts, id_base, it);
  ct_lexpr_proc(nd_qs.qs_q.q_contingut, id_base, it, pos);
  if is_valid(it) then
    ERROR:= true;
    missatges_menys_arguments_proc(pos, id_base);
    return;
  end if;

end ct_qs_proc;







procedure ct_lexpr_proc(nd_lexpr: in out pnode; id_base: in id_nom;it: in out iterador_arg;
                        pos: in out posicio) is
  desc_arg, desc_tipus_arg: descripcio;
  tsb_expr: tipus_subjacent;
  id_texpr, id_arg, id_tipus_arg: id_nom;
  esvar: boolean;
begin
  if nd_lexpr.lexpr_cont.tn /= nd_null then
    ct_lexpr_proc(nd_lexpr.lexpr_cont, id_base, it, pos);
  end if;

  ct_expr(nd_lexpr.lexpr_expr, id_texpr, tsb_expr, esvar, pos);
  if not is_valid(it) then
    ERROR:=true;
    missatges_massa_arguments_proc(pos, id_base);
    return;
  end if;
```

```
  get(ts, it, id_arg, desc_arg);
  case desc_arg.td is
    when dvar =>
      if not esvar then
        ERROR:= true;
        missatges_arg_mode(pos, id_arg);
      end if;
      id_tipus_arg:= desc_arg.tv;

    when dargc =>
      id_tipus_arg:= desc_arg.ta;

    when others =>
      ERROR:= true;
      missatges_ct_error_intern((fila=>31, columna=>20), "ct_lexpr_proc");
      return;
  end case;

   if id_texpr = null_id then
    if id_tipus_arg /= null_id then
      desc_tipus_arg:= get(ts, id_tipus_arg);
      if desc_tipus_arg.dt.tsb /= tsb_expr then
        ERROR:= true;
        missatges_tipus_incosistent_lit(pos, id_tipus_arg, tsb_expr);
      end if;
    else
      if tsb_expr /= tsb_nul then
        ERROR:=true;
        missatges_tipus_incosistent_lit(pos, id_tipus_arg, tsb_expr);
      end if;
    end if;
  else
    if id_tipus_arg /= id_texpr then
      ERROR:= true;
      missatges_tipus_incosistent_id(pos, id_tipus_arg, id_texpr);
    end if;
  end if;

  next(ts, it);

end ct_lexpr_proc;


procedure ct_qs(nd_qs: in out pnode; id_base: in id_nom; id_tipus: in out id_nom;
                pos: in out posicio) is
begin
    if nd_qs.qs_qs.tn /= nd_null then
      ct_qs(nd_qs.qs_qs, id_base, id_tipus, pos);
    end if;

    ct_q(nd_qs.qs_q, id_base, id_tipus, pos);
end ct_qs;


procedure ct_q(nd_q: in out pnode; id_base: in id_nom; id_tipus: in out id_nom;
               pos: in out posicio) is
  desc_tipus, desc_camp: descripcio;
  id_camp: id_nom;
  it: iterador_index;
  p: pnode;
begin
  desc_tipus:= get(ts, id_tipus);
  case nd_q.q_contingut.tn is
    when nd_id => --R.id
      if desc_tipus.dt.tsb /= tsb_rec then
        ERROR:= true;
        missatges_no_record(pos, id_tipus);
        return;
      end if;

      pos:= nd_q.q_contingut.id_pos;
      id_camp:= nd_q.q_contingut.id_id;
      desc_camp:= get_camp(ts, id_tipus, id_camp);
      if desc_camp.td /= dcamp then
        ERROR:= true;
        missatges_camp_no_record(pos, id_tipus, id_camp);
        return;
      end if;
      id_tipus:= desc_camp.tcmp;
```

```
      if not ERROR then
        p:= new node(nd_rec);
        nova_var_const(nv, tv, valor(desc_camp.dcmp), tsb_ent, p.rec_td);
      end if;

    when nd_lexpr => --R(E)
      if desc_tipus.dt.tsb /= tsb_arr then
        ERROR:= true;
        missatges_no_array(pos, id_tipus);
        return;
      end if;
      first(ts, id_tipus, it);
      ct_lexpr_array(nd_q.q_contingut, id_base, id_tipus, it, pos);

      if is_valid(it) then
        missatges_menys_indexos_array(pos, id_tipus);
      end if;
      id_tipus:= desc_tipus.dt.tcomp;

      if not ERROR then
        p:= new node(nd_arry);
        -- a ct_lexpr_array haurem anat penjant els nodes modificats.
        p.arry_lexpr:= nd_q.q_contingut;
        nova_var_const(nv, tv, valor(desc_tipus.dt.b), tsb_ent, p.arry_tb);
        desc_tipus:= get(ts,desc_tipus.dt.tcomp);
        nova_var_const(nv, tv, valor(desc_tipus.dt.ocup), tsb_ent, p.arry_tw);
      end if;

    when others =>
      ERROR:= true;
      missatges_ct_error_intern((fila=> 66, columna=>20), "ct_q");
  end case;
  if not ERROR then
    nd_q:= p;
  end if;
end ct_q;

procedure ct_lexpr_array(nd_lexpr: in out pnode; id_base: in id_nom; id_tipus: in out id_nom;
                         it: in out iterador_index; pos: in out posicio) is
  desc_index, desc_tipus_idx: descripcio;
  tsb_expr: tipus_subjacent;
  id_texpr: id_nom;
  esvar: boolean;
  p: pnode;
begin
  if nd_lexpr.lexpr_cont.tn /= nd_null then
    ct_lexpr_array(nd_lexpr.lexpr_cont, id_base, id_tipus, it, pos);
  end if;

  ct_expr(nd_lexpr.lexpr_expr, id_texpr, tsb_expr, esvar, pos);
  if not is_valid(it) then
    ERROR:= true;
    missatges_massa_indexos_array(pos, id_tipus);
    return;
  end if;

  desc_index:= get(ts, it);
  if id_texpr = null_id then
    desc_tipus_idx:= get(ts, desc_index.tind);
    if desc_tipus_idx.dt.tsb /= tsb_expr then
      ERROR:= true;
      missatges_tipus_incosistent_lit(pos, desc_index.tind, tsb_expr);
    end if;
  else
    if id_texpr /= desc_index.tind then
      ERROR:= true;
      missatges_tipus_incosistent_id(pos, desc_index.tind, id_texpr);
    end if;
  end if;
  next(ts, it);
  if not ERROR then
    desc_tipus_idx:= get(ts, desc_index.tind);
    p:= new node(nd_lexpr_arry);
    p.lexpra_cont:= nd_lexpr.lexpr_cont;
    p.lexpra_expr:= nd_lexpr.lexpr_expr;
    nova_var_const(nv, tv, valor(desc_tipus_idx.dt.lsup - desc_tipus_idx.dt.linf + 1),
                   tsb_ent, p.lexpra_tu);
    nd_lexpr:= p;
  end if;
end ct_lexpr_array;
```

```
procedure ct_expr(nd_expr: in out pnode; id_texpr: out id_nom; tsb_expr: out tipus_subjacent;
                  esvar: out boolean; pos: in out posicio) is
begin
  case nd_expr.expr_e.tn is
    when nd_and | nd_or =>
      ct_e(nd_expr.expr_e, id_texpr, tsb_expr, esvar, pos);

    when nd_eop =>
      ct_eop(nd_expr.expr_e, id_texpr, tsb_expr, esvar, pos);

    when others =>
      ERROR:= true;
      missatges_ct_error_intern((fila=>11, columna=>20), "ct_expr");

  end case;
end ct_expr;


procedure ct_e(nd_e: in out pnode; id_texpr: out id_nom; tsb_expr: out tipus_subjacent;
               esvar: out  boolean; pos: in out posicio) is
  id_tipus1, id_tipus2: id_nom;
  tsb1, tsb2: tipus_subjacent;
  esvar1, esvar2: boolean;
  pos1, pos2: posicio:= (0, 0);
  error: boolean;
begin
  if nd_e.e_ope.tn = nd_and or else nd_e.e_ope.tn = nd_or then
    ct_e(nd_e.e_ope, id_tipus1, tsb1, esvar1, pos1);
  else
    ct_eop(nd_e.e_ope, id_tipus1, tsb1, esvar1, pos1);
  end if;

  ct_eop(nd_e.e_opd, id_tipus2, tsb2, esvar2, pos2);

  tipus_compatible(id_tipus1, id_tipus2, tsb1, tsb2, id_texpr, tsb_expr, error);
  if error then
    ERROR:= true;
    missatges_expressions_incompatibles(pos1, id_tipus1, id_tipus2, tsb1, tsb2);
  end if;

  if tsb1 /= tsb_bool then
    ERROR:= true;
    missatges_log_operador(pos1, tsb1);
  end if;

  pos:= pos1;
  esvar:= false;
end ct_e;


procedure ct_eop(nd_e: in out pnode; id_texpr: out id_nom; tsb_expr: out tipus_subjacent;
                 esvar: out boolean; pos: in out posicio) is
begin
  case nd_e.eop_operand is
    when major | menor | menorigual | majorigual | igual | diferent  =>
      ct_eop_op_rel(nd_e, id_texpr, tsb_expr, esvar, pos);

    when sum | res | prod | quoci | modul =>
      ct_eop_arit(nd_e, id_texpr, tsb_expr, esvar, pos, nd_e.eop_operand);

    when neg_log =>
      ct_eop_neg_log(nd_e, id_texpr, tsb_expr, esvar, pos);

    when neg_alg =>
      ct_eop_neg_arit(nd_e, id_texpr, tsb_expr, esvar, pos);

    when nul =>
      ct_et(nd_e.eop_opd, id_texpr, tsb_expr, esvar, pos);

    when others =>
      ERROR:= true;
      missatges_ct_error_intern((fila=>27, columna=>20), "ct_eop");
  end case;
end ct_eop;
```

```
procedure ct_eop_op_rel(nd_e: in out pnode; id_texpr: out id_nom;
                        tsb_expr: out tipus_subjacent; esvar: out boolean;
                        pos: in out posicio) is
  id_tipus1, id_tipus2: id_nom;
  tsb1, tsb2: tipus_subjacent;
  esvar1, esvar2: boolean;
  pos1, pos2: posicio:= (0, 0);
  error: boolean;
begin
  ct_eop(nd_e.eop_ope, id_tipus1, tsb1, esvar1, pos1);
  ct_eop(nd_e.eop_opd, id_tipus2, tsb2, esvar2, pos2);

  tipus_compatible(id_tipus1, id_tipus2, tsb1, tsb2, id_texpr, tsb_expr, error);

  if tsb1 > tsb_ent then
    ERROR:= true;
    missatges_operador_tipus(pos1, tsb1, nd_e.eop_operand);
  end if;

  pos:= pos1;
  id_texpr:= null_id;
  tsb_expr:= tsb_bool;
  esvar:= false;
end ct_eop_op_rel;



procedure ct_eop_arit(nd_e: in out pnode; id_texpr: out id_nom;
                      tsb_expr: out tipus_subjacent; esvar: out boolean;
                      pos: in out posicio; op: in operand ) is
  id_tipus1, id_tipus2: id_nom;
  tsb1, tsb2: tipus_subjacent;
  esvar1, esvar2: boolean;
  pos1, pos2: posicio:= (0, 0);
  error: boolean;
begin
  ct_eop(nd_e.eop_ope, id_tipus1, tsb1, esvar1, pos1);
  ct_eop(nd_e.eop_opd, id_tipus2, tsb2, esvar2, pos2);
  tipus_compatible(id_tipus1, id_tipus2, tsb1, tsb2, id_texpr, tsb_expr, error);
  if error then
    ERROR:= true;
    missatges_expressions_incompatibles(pos1, id_tipus1, id_tipus2,tsb1, tsb2);
  end if;

  if tsb1 /= tsb_ent then
    ERROR:= true;
    missatges_operador_tipus(pos1, tsb1, op);
  end if;

  pos:= pos1;
  esvar:= false;
end ct_eop_arit;



procedure ct_eop_neg_log(nd_e: in out pnode; id_texpr: out id_nom;
                         tsb_expr: out tipus_subjacent; esvar: out boolean;
                         pos: in out posicio) is
  id_tipus: id_nom;
  tsb: tipus_subjacent;
  esvar1: boolean;
begin
  ct_et(nd_e.eop_opd, id_tipus, tsb, esvar1, pos);
  if tsb /= tsb_bool then
    ERROR:= true;
    missatges_operador_tipus(pos, tsb, neg_log);
  end if;
  id_texpr:= id_tipus;
  tsb_expr:= tsb;
  esvar:= false;
end ct_eop_neg_log;
```

```
procedure ct_eop_neg_arit(nd_e: in out pnode; id_texpr: out id_nom;
                          tsb_expr: out tipus_subjacent; esvar: out boolean;
                          pos: in out posicio) is
  id_tipus: id_nom;
  tsb: tipus_subjacent;
  esvar1: boolean;
begin
  ct_et(nd_e.eop_opd, id_tipus, tsb, esvar1, pos);
  if tsb /= tsb_ent then
    ERROR:= true;
    missatges_operador_tipus(pos, tsb, neg_alg);
  end if;
  id_texpr:= id_tipus;
  tsb_expr:= tsb;
  esvar:= false;
end ct_eop_neg_arit;


procedure ct_et(nd_e: in out pnode; id_texpr: out id_nom; tsb_expr: out tipus_subjacent;
                esvar: out  boolean; pos: in out posicio) is
begin
  case nd_e.et_cont.tn is
    when nd_ref =>
      ct_et_ref(nd_e.et_cont, id_texpr, tsb_expr, esvar, pos);

    when nd_expr =>
      ct_expr(nd_e.et_cont, id_texpr, tsb_expr, esvar, pos);

    when nd_lit =>
      ct_et_lit(nd_e.et_cont, id_texpr, tsb_expr, esvar, pos);

    when others =>
      ERROR:= true;
      missatges_ct_error_intern((fila=>18, columna=>16), "ct_et");
  end case;
end ct_et;


procedure ct_et_ref(nd_ref: in out pnode; id_texpr: out id_nom;
                    tsb_expr: out tipus_subjacent; esvar: out boolean;
                    pos: in out posicio) is
  id_ref, id_tipus_ref: id_nom;
  desc_ref, desc_tipus_ref: descripcio;
begin
  ct_ref(nd_ref, id_ref, id_tipus_ref, pos);

  id_texpr:= id_tipus_ref;

  desc_tipus_ref:= get(ts, id_tipus_ref);
  tsb_expr:= desc_tipus_ref.dt.tsb;

  desc_ref:= get(ts, id_ref);
  case desc_ref.td is
    when dvar => esvar:= true;
    when dconst => esvar:= false;
    when dargc => esvar:= false;
    when others =>
      ERROR:= true;
      missatges_ct_error_intern((fila=>18, columna=>16), "ct_et_ref");
  end case;
end ct_et_ref;


procedure ct_et_lit(nd_lit: in out pnode; id_texpr: out id_nom;
                    tsb_expr: out tipus_subjacent; esvar: out boolean;
                    pos: in out posicio) is
begin
  id_texpr:= null_id;
  pos:= nd_lit.lit_pos;
  tsb_expr:= nd_lit.lit_tipus;
  esvar:= false;
end ct_et_lit;
```

```
  procedure tipus_compatible(id_tipus1, id_tipus2: in id_nom; tsb1, tsb2: in tipus_subjacent;
                             id_texp: out id_nom; tsb_exp: out tipus_subjacent;
                             error: out boolean) is
  begin
    error:=false;
    if id_tipus1 = null_id and id_tipus2 = null_id then
      if tsb1 /= tsb2 then error:= true; end if;
      id_texp:= null_id; tsb_exp:= tsb1;

    elsif id_tipus1 /= null_id and id_tipus2 = null_id then
      if tsb1 /= tsb2 then error:= true; end if;
      id_texp:= id_tipus1; tsb_exp:= tsb1;

    elsif id_tipus1 = null_id and id_tipus2 /= null_id then
      if tsb1 /= tsb2 then error:= true; end if;
      id_texp:= id_tipus2; tsb_exp:= tsb2;

    elsif id_tipus1 /= null_id and id_tipus2 /= null_id then
      if tsb1 /= tsb2 then error:= true; end if;
      id_texp:= id_tipus1; tsb_exp:= tsb1;

    end if;
  end tipus_compatible;

end semantica.c_tipus;
```

```
package semantica.g_codi_int is

  procedure prepara_g_codi_int(nomf: in String; c,f: in num_var);
  procedure gen_codi_int;

end semantica.g_codi_int;



--BODY
with Ada.Strings.Unbounded; use Ada.Strings.Unbounded;
with decls.d_tnoms;
with decls.d_descripcio; use decls.d_descripcio;
with decls.d_c3a; use decls.d_c3a;
package body semantica.g_codi_int is

  use Instruccio_IO;
  use Pila_Procediments;

  f3a:  Instruccio_IO.File_Type;
  f3as: Ada.Text_IO.File_Type;
  nf: Unbounded_String;

  fals: num_var;
  cert: num_var;

  procedure gc_proc(nd_proc: in pnode);
  procedure gc_cproc(nd_cproc: in pnode);


  procedure gc_decls(nd_decls: in pnode);


  procedure gc_sents(nd_sents: in pnode);
  procedure gc_sent(nd_sent: in pnode);
  procedure gc_siter(nd_siter: in pnode);
  procedure gc_scond(nd_scond: in pnode);
  procedure gc_scrida(nd_cproc: in pnode);
  procedure gc_scrida_args(nod_lexpr: in pnode);
  procedure gc_sassign(nd_sassign: in pnode);


  procedure gc_ref(nd_ref: in pnode; r: out num_var; d: out num_var);
  procedure gc_ref_id(nd_id: in pnode; r: out num_var; dc: out despl;
                      dv: out num_var);
  procedure gc_ref_qs(nd_qs: in pnode; dc: in out despl; dv: in out num_var);
  procedure gc_ref_lexpr(nd_lexpr: in pnode; desp: in out num_var);


  procedure gc_expressio(nd_expr: in pnode; r: out num_var; d: out num_var);
  procedure gc_and(nod_and: in pnode; r: out num_var; d: out num_var);
  procedure gc_or(nod_or: in pnode; r: out num_var; d: out num_var);
  procedure gc_eop(nd_eop: in pnode; r: out num_var; d: out num_var);
  procedure gc_et(nd_et: in pnode; r: out num_var; d: out num_var);


  procedure prepara_g_codi_int(nomf: in String; c,f: in num_var) is
  begin
    cert:= c;
    fals:= f;
    nf:= To_Unbounded_String(nomf);
  end prepara_g_codi_int;


  procedure gen_codi_int is
  begin
    Create(f3a, Out_File, To_String(nf)&".c3a");
    Create(f3as, Out_File, To_String(nf)&".c3as");

    if root.p.tn /= nd_null then
      buida(pproc);
      gc_proc(root.p);
    end if;

    Close(f3as);
    Close(f3a);
  end gen_codi_int;
```

```
procedure genera(i3a: in instr_3a) is
begin
  Instruccio_IO.Write(f3a, To_i3a_bin(i3a));
  Ada.Text_IO.Put_Line(f3as, Imatge(i3a, tv, tp));
end genera;


procedure desref(r: in num_var; d: in num_var; t: out num_var) is
begin
  if d = null_nv then
   t:= r;
  else
   -- no tinc del tot clar aquest ocup_ent, simplifica les coses pero
   -- es una tudada de memoria
   nova_var(nv, tv, tp, cim(pproc), ocup_ent, t);
   genera(Value(cons_idx, t, r, d));
  end if;
end desref;


procedure gc_proc(nd_proc: in pnode) is
  p: pnode renames nd_proc;
begin
  empila(pproc, p.proc_cproc.cproc_np);

  if p.proc_decls.tn /= nd_null then
    gc_decls(p.proc_decls);
  end if;

  gc_cproc(p.proc_cproc);

  gc_sents(p.proc_sents);

  genera(Value(rtn, cim(pproc)));
  desempila(pproc);
end gc_proc;


procedure gc_cproc(nd_cproc: in pnode) is
  p: pnode renames nd_cproc;
begin
  genera(Value(etiq, c_etiq_proc(tp, cim(pproc))));
  genera(Value(pmb, cim(pproc)));
end gc_cproc;


procedure gc_decls(nd_decls: in pnode) is
  p: pnode renames nd_decls;
begin
  if p.decls_decls.tn /= nd_null then
    gc_decls(p.decls_decls);
  end if;
  -- nomes cal generar codi per als procediments
  if p.decls_decl.decl_real.tn = nd_proc then
    gc_proc(p.decls_decl.decl_real);
  end if;
end gc_decls;


procedure gc_sents(nd_sents: in pnode) is
begin
  if nd_sents.tn /= nd_null then
    gc_sent(nd_sents.sents_cont);
  end if;
end gc_sents;
```

```
procedure gc_sent(nd_sent: in pnode) is
  p: pnode;
begin
  p:= nd_sent;
  if p.snb_snb.tn /= nd_null then
    gc_sent(p.snb_snb);
  end if;
  p:= p.snb_sent.sent_sent;
  case p.tn is
    when nd_siter =>
      gc_siter(p);
    when nd_scond =>
      gc_scond(p);
    when nd_scrida =>
      gc_scrida(p);
    when nd_sassign =>
      gc_sassign(p);
    when others =>
      null;
  end case;
end gc_sent;


procedure gc_siter(nd_siter: in pnode) is
  p: pnode renames nd_siter;
  ei,ef: num_etiq;
  r,t,d: num_var;
begin
  if p.siter_sents.tn /= nd_null then
    nova_etiq(ne, ei);
    genera(Value(etiq, ei));

    gc_expressio(p.siter_expr, r, d);
    desref(r, d, t); -- Bastaria amb un byte per codificar -1..0

    nova_etiq(ne, ef);
    genera(Value(ieq_goto, ef, t, fals));

    gc_sents(p.siter_sents);

    genera(Value(go_to, ei));
    genera(Value(etiq, ef));
  end if;
end gc_siter;

procedure gc_scond(nd_scond: in pnode) is
  p: pnode renames nd_scond;
  ef,efi: num_etiq;
  r,t,d: num_var;
begin
  if p.scond_sents.tn /= nd_null or p.scond_esents.tn /= nd_null then
    gc_expressio(p.scond_expr, r, d);
    desref(r, d, t);

    nova_etiq(ne, ef);
    genera(Value(ieq_goto, ef, t, fals));

    gc_sents(p.scond_sents);
    if p.scond_esents.tn /= nd_null then
      nova_etiq(ne, efi);
      genera(Value(go_to, efi));
      genera(Value(etiq, ef));
      gc_sents(p.scond_esents);
      -- codi esperat:
      -- if t=fals goto efals
      --  sents_if
      --  goto efi
      -- efals: skip
      --  sents_else
      -- efi: skip
    else
      efi:= ef; --tefi:= tef;
      -- codi esperat:
      -- if t=fals goto efi
      --  sents_if
      -- efi: skip
    end if;
    genera(Value(etiq, efi));
  end if;
end gc_scond;
```

```
procedure gc_scrida(nd_cproc: in pnode) is
  p: pnode;
  d: descripcio;
begin
  p:= nd_cproc.scrida_ref;
  if p.ref_qs.tn /= nd_null then
    gc_scrida_args(p.ref_qs.qs_q.q_contingut);
  end if;

  if p.ref_id.tn = nd_id then -- stdio call
    d:= get(ts, p.ref_id.id_id);
    genera(Value(call, d.np));
  else
    genera(Value(call, p.ref_id.iproc_np));
  end if;
end gc_scrida;


procedure gc_scrida_args(nod_lexpr: in pnode) is
  p: pnode renames nod_lexpr;
  r,d: num_var;
begin
  if p.lexpr_cont.tn /= nd_null then
    gc_scrida_args(p.lexpr_cont);
  end if;

  gc_expressio(p.lexpr_expr, r, d);

  if d = null_nv then
    genera(Value(params, r));
  else
    genera(Value(paramc, r, d));
  end if;
end gc_scrida_args;


procedure gc_sassign(nd_sassign: in pnode) is
  p: pnode renames nd_sassign;
  r,r1,t,d,d1: num_var;
begin
  gc_ref(p.sassign_ref, r, d);
  gc_expressio(p.sassign_expr, r1, d1);

  if d = null_nv then
    if d1 = null_nv then
      genera(Value(cp, r, r1));
      -- r:= r1
    else
      genera(Value(cons_idx, r, r1, d1));
      -- r:= r1[d1]
    end if;
  else
    if d1 = null_nv then
      genera(Value(cp_idx, r, d, r1));
      -- r[d]:= r1
    else
      desref(r1, d1, t);
      genera(Value(cp_idx, r,  d, t));
      -- t:= r1[d1]
      -- r[d]:= t
    end if;
  end if;
end gc_sassign;
```

```
procedure gc_ref(nd_ref: in pnode; r: out num_var; d: out num_var) is
  p: pnode renames nd_ref;
  dc: despl;
  t,t1,dv: num_var;
begin
  gc_ref_id(p.ref_id, r, dc, dv);
  if p.ref_qs.tn /= nd_null then
    gc_ref_qs(p.ref_qs, dc, dv);
  end if;

  if dc = 0 and then dv = 0 then
    -- r = r
    d:= null_nv;
  elsif dc = 0 and then dv /= 0 then
    -- r = r(dv)
    d:= dv;
  elsif dc /= 0 and then dv = 0 then
    nova_var_const(nv, tv, valor(dc), tsb_ent, t);
    -- r:= r.dc
    d:= t;
  else -- dc /= 0 and dv /= 0
    nova_var_const(nv, tv, valor(dc), tsb_ent, t);
    nova_var(nv, tv, tp, cim(pproc), ocup_ent, t1);
    genera(Value(sum, t1, t, dv));
    -- r:= r.dc(dv)
    d:= t1;
  end if;
end gc_ref;


procedure gc_ref_id(nd_id: in pnode; r: out num_var; dc: out despl;
                    dv: out num_var) is
  p: pnode renames nd_id;
begin
  r:= p.var_nv;
  dc:= 0;
  dv:= null_nv;
end gc_ref_id;


procedure gc_ref_qs(nd_qs: in pnode; dc: in out despl; dv: in out num_var)
is
  p: pnode renames nd_qs;
  t,t1,t2: num_var;
  desp: num_var:= null_nv;
begin
  if p.qs_qs.tn /= nd_null then
    gc_ref_qs(p.qs_qs, dc, dv);
  end if;

  if p.qs_q.tn = nd_rec then
    dc:= dc + despl(c_val_const(tv, p.qs_q.rec_td));
  else -- p.qs_q.tn = nd_arry
    gc_ref_lexpr(p.qs_q.arry_lexpr, desp);

    nova_var(nv, tv, tp, cim(pproc), ocup_ent, t);
    genera(Value(res, t, desp, p.qs_q.arry_tb));

    nova_var(nv, tv, tp, cim(pproc), ocup_ent, t1);
    genera(Value(mul, t1, t, p.qs_q.arry_tw));

    if dv = null_nv then
      dv:= t1;
    else
      nova_var(nv, tv, tp, cim(pproc), ocup_ent, t2);
      genera(Value(sum, t2, t1, dv));
      dv:= t2;
    end if;
  end if;
end gc_ref_qs;
```

```
procedure gc_ref_lexpr(nd_lexpr: in pnode; desp: in out num_var) is
  p: pnode renames nd_lexpr;
  r,d,t,t1,te: num_var;
begin
  if p.lexpra_cont.tn /= nd_null then
    gc_ref_lexpr(p.lexpra_cont, desp);
  end if;

  if desp /= 0 then
    nova_var(nv, tv, tp, cim(pproc), ocup_ent, t);
    genera(Value(mul, t, desp, p.lexpra_tu));

    gc_expressio(p.lexpra_expr, r, d);
    desref(r, d, te);

    nova_var(nv, tv, tp, cim(pproc), ocup_ent, t1);
    genera(Value(sum, t1, t, te));
    desp:= t1;
  else
    nova_var(nv, tv, tp, cim(pproc), ocup_ent, t);

    gc_expressio(p.lexpra_expr, r, d);
    desref(r, d, te);

    genera(Value(cp, t, te, null_nv));
    desp:= t;
  end if;
end gc_ref_lexpr;


procedure gc_expressio(nd_expr: in pnode; r: out num_var; d: out num_var)
is
  p: pnode renames nd_expr;
begin
  case p.expr_e.tn is
    when nd_and =>
      gc_and(p.expr_e, r, d);
      --ocup:= ocup_bool;

    when nd_or =>
      gc_or(p.expr_e, r, d);
      --ocup:= ocup_bool;

    when nd_eop | nd_op_rel =>
      gc_eop(p.expr_e, r, d);
      --ocup:= max(ocup_[terme1..termeN])

    when others =>
      null;
      -- Comprovació de tipus

  end case;
end gc_expressio;


procedure gc_and(nod_and: in pnode; r: out num_var; d: out num_var) is
  p: pnode renames nod_and;
  t,t1,t2: num_var;
  r1,d1: num_var;
begin
  if p.e_ope.tn = nd_and then
    gc_and(p.e_ope, r1, d1);
  else
    gc_eop(p.e_ope, r1, d1);
  end if;
  -- bastaria amb un char/bit -1..0 per guardar el resultat
  desref(r1, d1, t1);

  gc_eop(p.e_opd, r1, d1);
  -- bastaria amb un char/bit -1..0 per guardar el resultat
  desref(r1, d1, t2);

  -- bastaria amb un char/bit -1..0 per guardar el resultat
  nova_var(nv, tv, tp, cim(pproc), ocup_ent, t);
  genera(Value(op_and, t, t1, t2));
  r:= t;
  d:= null_nv;
end gc_and;
```

```
procedure gc_or(nod_or: in pnode; r: out num_var; d: out num_var) is
  p: pnode renames nod_or;
  t,t1,t2: num_var;
  r1,d1: num_var;
begin
  if p.e_ope.tn = nd_or then
    gc_or(p.e_ope, r1, d1);
  else
    gc_eop(p.e_ope, r1, d1);
  end if;
  desref(r1, d1, t1);

  gc_eop(p.e_opd, r1, d1);
  desref(r1, d1, t2);

  nova_var(nv, tv, tp, cim(pproc), ocup_ent, t);
  genera(Value(op_or, t, t1, t2));
  r:= t;
  d:= null_nv;
end gc_or;


procedure gc_eop(nd_eop: in pnode; r: out num_var; d: out num_var) is
  p: pnode renames nd_eop;
  t,t1,t2: num_var;
  r1,d1: num_var;
begin
  if p.eop_operand = nul then
    gc_et(p.eop_opd, r, d);
  elsif p.eop_operand = neg_alg or p.eop_operand = neg_log then
    nova_var(nv, tv, tp, cim(pproc), ocup_ent, t);
    gc_et(p.eop_opd, r1, d1);
    desref(r1, d1, t1);

    if p.eop_operand = neg_alg then
      genera(Value(neg, t, t1, null_nv));
    else
      genera(Value(op_not, t, t1, null_nv));
    end if;
    r:= t;
    d:= null_nv;
  else
    gc_eop(p.eop_ope, r1, d1);
    desref(r1, d1, t1);

    gc_eop(p.eop_opd, r1, d1);
    desref(r1, d1, t2);

    nova_var(nv, tv, tp, cim(pproc), ocup_ent, t);
    case p.eop_operand is
      when major =>
        genera(Value(gt, t, t1, t2));

      when majorigual =>
        genera(Value(ge, t, t1, t2));

      when igual =>
        genera(Value(eq, t, t1, t2));

      when diferent =>
        genera(Value(neq, t, t1, t2));

      when menorigual =>
        genera(Value(le, t, t1,  t2));

      when menor =>
        genera(Value(lt, t, t1, t2));

      when sum =>
        genera(Value(sum, t, t1, t2));

      when res =>
        genera(Value(res, t, t1, t2));

      when prod =>
        genera(Value(mul, t, t1, t2));

      when quoci =>
        genera(Value(div, t, t1, t2));
```

```
        when modul =>
          genera(Value(modul, t, t1, t2));

        when others =>
          null;

      end case;
      r:= t;
      d:= null_nv;
    end if;
  end gc_eop;


  procedure gc_et(nd_et: in pnode; r: out num_var; d: out num_var) is
    p: pnode renames nd_et;
    t: num_var;
  begin
    case p.et_cont.tn is
      when nd_ref =>
        gc_ref(p.et_cont, r, d);

      when nd_expr =>
        gc_expressio(p.et_cont, r, d);

      when nd_lit =>
        case p.et_cont.lit_tipus is
          when tsb_ent | tsb_car | tsb_bool | tsb_nul =>
            nova_var_const(nv, tv, p.et_cont.lit_val,
                           p.et_cont.lit_tipus, t);

          when others =>
            null;

        end case;
        r:= t;
        d:= null_nv;
      when others =>
        null;

    end case;
  end gc_et;
end semantica.g_codi_int;
```

```
package semantica.g_codi_ass is

  procedure gen_codi_ass;
  procedure prepara_g_codi_ass(nomf: in String);

end semantica.g_codi_ass;


--BODY
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Strings.Unbounded; use Ada.Strings.Unbounded;
with Ada.Strings.Fixed; use Ada.Strings.Fixed; --El trim
with decls.d_tnoms;
with decls.d_c3a; use decls.d_c3a;
with decls.d_descripcio; use decls.d_descripcio;
with semantica; use semantica;
package body semantica.g_codi_ass is
  use Instruccio_IO;
  use Pila_Procediments;

  Output: Ada.Text_IO.File_Type;
  Input: Instruccio_IO.File_Type;
  newline: String(1..1):=(1=>ASCII.LF); --new line

  nf: Unbounded_String;

  type registre is (
    eax,
    ebx,
    ecx,
    edx,
    esi,
    edi,
    ebp,
    esp
  );

  --Definicions
  function ga_llegir return instr_3a;
  procedure ga_escriure(text: in String);

  procedure generacio_assemblador;
  procedure init_memoria;


  function ga_load(nv: in num_var; r: in registre) return String;
  function ga_load_constant(nv: in num_var; r: in registre) return String;
  function ga_load_var_local(nv: in num_var; r: in registre) return String;
  function ga_load_param_local(nv: in num_var; r: in registre) return String;
  function ga_load_var_global(nv: in num_var; r: in registre) return String;
  function ga_load_param_global(nv: in num_var; r: in registre)
  return String;


  function ga_store(nv: in num_var; r: in registre) return String;
  function ga_store_var_local(nv: in num_var; r: in registre) return String;
  function ga_store_param_local(nv: in num_var; r: in registre)
  return String;
  function ga_store_var_global(nv: in num_var; r: in registre) return String;
  function ga_store_param_global(nv: in num_var; r: in registre)
  return String;


  function ga_load_address(nv: in num_var; r: in registre) return String;
  function ga_load_address_constant(nv: in num_var; r: in registre)
  return String;
  function ga_load_address_var_local(nv: in num_var; r: in registre)
  return String;
  function ga_load_address_param_local(nv: in num_var; r: in registre)
  return String;
  function ga_load_address_var_global(nv: in num_var; r: in registre)
  return String;
  function ga_load_address_param_global(nv: in num_var; r: in registre)
  return String;
```

```
procedure ga_cp(i3a: in instr_3a);
procedure ga_cons_idx(i3a: in instr_3a);
procedure ga_cp_idx(i3a: in instr_3a);


procedure ga_sum(i3a: in instr_3a);
procedure ga_res(i3a: in instr_3a);
procedure ga_mul(i3a: in instr_3a);
procedure ga_div(i3a: in instr_3a);
procedure ga_modul(i3a: in instr_3a);
procedure ga_neg(i3a: in instr_3a);
procedure ga_op_not(i3a: in instr_3a);
procedure ga_op_and(i3a: in instr_3a);
procedure ga_op_or(i3a: in instr_3a);


function ga_etiq(e: in num_etiq) return String;
function ga_goto(e: in num_etiq) return String;
procedure ga_goto(i3a: in instr_3a);
procedure ga_etiq(i3a: in instr_3a);
procedure ga_ieq_goto(i3a: in instr_3a);

procedure ga_gt(i3a: in instr_3a);
procedure ga_ge(i3a: in instr_3a);
procedure ga_eq(i3a: in instr_3a);
procedure ga_neq(i3a: in instr_3a);
procedure ga_le(i3a: in instr_3a);
procedure ga_lt(i3a: in instr_3a);


procedure ga_pmb(i3a: in instr_3a);
procedure ga_rtn(i3a: in instr_3a);
procedure ga_call(i3a: in instr_3a);
procedure ga_paramc(i3a: in instr_3a);
procedure ga_params(i3a: in instr_3a);


function Value(*) return String is
begin
  -- omesa per conveniència
end Value;


procedure gen_codi_ass is
begin
  Open(File=>Input, Mode=>In_File, Name=> To_String(nf) & ".c3a");
  Create(File=>Output, Mode=>Out_File, Name=> To_String(nf) & ".s");

  buida(pproc);
  calcul_desplacaments(tv, nv, tp, np);
  generacio_assemblador;

  Close(Input);
  Close(Output);
end gen_codi_ass;


procedure prepara_g_codi_ass(nomf: in String) is
begin
  nf:= To_Unbounded_String(nomf);
end prepara_g_codi_ass;


function ga_llegir return instr_3a is
  inst: instr_3a_bin;
begin
  Instruccio_IO.Read(Input, Item=> inst);
  return To_i3a(inst);
end ga_llegir;

procedure ga_escriure(text: in String) is
begin
  Put(File=> Output, Item=> text);
end ga_escriure;
```

```
procedure generacio_assemblador is
  inst: instr_3a;
begin
  init_memoria;
  while not End_Of_File(Input) loop
    inst:=ga_llegir;
    case c_tipus(inst) is
      -- per cada tipus d'instrucció, crida al pertinent procediment
      -- i afegeix un newline a continuació. S'ha omés per conveniència
    end case;
  end loop;
end generacio_assemblador;


--Inicialitzar memoria
procedure init_memoria is
begin
  ga_escriure(".section  .bss" & newline);
  ga_escriure("  .comm DISP, 100" & newline);

  ga_escriure(".section .text" & newline);
  ga_escriure("  .global main" & newline);
end init_memoria;


--LOAD
function ga_load(nv: in num_var; r: in registre) return String is
begin
  if not es_var(tv, nv) then
    return ga_load_constant(nv, r);

  elsif c_prof_proc(tp, c_np_var(tv, nv)) = c_prof_proc(tp, cim(pproc))
  then
    if c_desp_var(tv, nv)<0 then
      return ga_load_var_local(nv, r);

    elsif c_desp_var(tv, nv)>0 then
      return ga_load_param_local(nv, r);
    end if;

  elsif c_prof_proc(tp, c_np_var(tv, nv)) < c_prof_proc(tp, cim(pproc))
  then
    if c_desp_var(tv, nv)<0 then
      return ga_load_var_global(nv, r);

    elsif c_desp_var(tv, nv)>0 then
      return ga_load_param_global(nv, r);
    end if;
  end if;
  return "";
end ga_load;


function ga_load_constant(nv: in num_var; r: in registre) return String
is
  reg: constant String:= registre'Image(r);
  valor_const: constant String:= Value(c_val_const(tv, nv));
begin
  return "  movl  $" & valor_const &", %" & reg & newline;
end ga_load_constant;


function ga_load_var_local(nv: in num_var; r: in registre) return String
is
  reg: constant String:= registre'Image(r);
  desplacament: constant String:= Value(c_desp_var(tv, nv));
begin
  return "  movl " & desplacament & "(%ebp), %" & reg & newline;
end ga_load_var_local;


function ga_load_param_local(nv: in num_var; r: in registre) return String
is
  reg: constant String:= registre'Image(r);
  desplacament: constant String:= Value(c_desp_var(tv, nv));
begin
  return  "  movl " & desplacament & "(%ebp), %esi" & newline
        & "  movl (%esi), %" & reg & newline;
end ga_load_param_local;
```

```
function ga_load_var_global(nv: in num_var; r: in registre) return String
is
  desp_disp: constant Integer:= 4*Value(c_prof_proc(tp, c_np_var(tv, nv)));
  desp_display: constant String:=Value(desp_disp);
  reg: constant String:= registre'Image(r);
  desplacament: constant String:= Value(c_desp_var(tv, nv));
begin
  return  "  movl $DISP, %esi" & newline
        & "  movl " & desp_display & "(%esi), %esi" & newline
        & "  movl " & desplacament & "(%esi), %" & reg & newline;
end ga_load_var_global;


function ga_load_param_global(nv: in num_var; r: in registre) return String
is
  desp_disp: constant Integer:= 4*Value(c_prof_proc(tp,c_np_var(tv,nv)));
  desp_display: constant String:=Value(desp_disp);
  reg: constant String:= registre'Image(r);
  desplacament: constant String:= Value(c_desp_var(tv, nv));
begin
  return  "  movl $DISP, %esi" & newline
        & "  movl " & desp_display & "(%esi), %esi" & newline
        & "  movl " & desplacament & "(%esi), %esi" & newline
        & "  movl (%esi), %" & reg & newline;
end ga_load_param_global;


--STORE
function ga_store(nv: in num_var; r: in registre) return String is
begin
  if c_prof_proc(tp, c_np_var(tv, nv)) = c_prof_proc(tp, cim(pproc))
  then
    if c_desp_var(tv, nv)<0 then
      return ga_store_var_local(nv, r );

    elsif c_desp_var(tv, nv)>0 then
      return ga_store_param_local(nv, r);
    end if;

  elsif c_prof_proc(tp, c_np_var(tv, nv)) < c_prof_proc(tp, cim(pproc))
  then
    if c_desp_var(tv, nv)<0 then
      return ga_store_var_global(nv, r);

    elsif c_desp_var(tv, nv)>0 then
      return ga_store_param_global(nv, r);
    end if;
  end if;
  return "";
end ga_store;


function ga_store_var_local(nv: in num_var; r: in registre) return String
is
  reg: constant String:= registre'Image(r);
  desplacament: constant String:= Value(c_desp_var(tv, nv));
begin
  return "  movl %" & reg & ", " & desplacament & "(%ebp)" & newline;
end ga_store_var_local;


function ga_store_param_local(nv: in num_var; r: in registre)
return String is
  reg: constant String:= registre'Image(r);
  desplacament: constant String:= Value(c_desp_var(tv, nv));
begin
  return "  movl " & desplacament & "(%ebp), %edi" & newline
       & "  movl %" & reg & ", (%edi)" & newline;
end ga_store_param_local;
```

```
function ga_store_var_global(nv: in num_var; r: in registre) return String
is
  desp_disp: constant Integer:= 4*Value(c_prof_proc(tp,c_np_var(tv,nv)));
  desp_display: constant String:=Value(desp_disp);
  reg: constant String:= registre'Image(r);
  desplacament: constant String:= Value(c_desp_var(tv, nv));
begin
  return  "  movl $DISP, %esi" & newline
        & "  movl " & desp_display & "(%esi), %edi" & newline
        & "  movl %" & reg & " ," & desplacament & "(%edi)" & newline;
end ga_store_var_global;


function ga_store_param_global(nv: in num_var; r: in registre)
return String is
  desp_disp: constant Integer:= 4*Value(c_prof_proc(tp, c_np_var(tv,nv)));
  desp_display: constant String:= Value(desp_disp);
  reg: constant String:= registre'Image(r);
  desplacament: constant String:= Value(c_desp_var(tv, nv));
begin
  return  "  movl $DISP, %esi" & newline
        & "  movl " & desp_display & "(%esi), %esi" & newline
        & "  movl " & desplacament & "(%esi), %edi" & newline
        & "  movl %" & reg & ", (%edi)" & newline;
end ga_store_param_global;


--LOAD ADDRESS
function ga_load_address(nv: in num_var; r: in registre) return String
is
begin
  if not es_var(tv, nv) then
    return ga_load_address_constant(nv, r);

  elsif c_prof_proc(tp, c_np_var(tv, nv)) = c_prof_proc(tp, cim(pproc))
  then
    if c_desp_var(tv, nv)<0 then
      return ga_load_address_var_local(nv, r);

    elsif c_desp_var(tv, nv)>0 then
      return ga_load_address_param_local(nv, r);
    end if;

  elsif c_prof_proc(tp, c_np_var(tv, nv)) < c_prof_proc(tp, cim(pproc))
  then
    if c_desp_var(tv, nv)<0 then
      return ga_load_address_var_global(nv, r);

    elsif c_desp_var(tv, nv)>0 then
      return ga_load_address_param_global(nv, r);
    end if;
  end if;
  return "";
end ga_load_address;


function ga_load_address_constant(nv: in num_var; r: in registre)
return String is
  ecx: num_etiq;
begin
  nova_etiq(ne, ecx);
  case c_tsb_const(tv, nv) is
    when tsb_ent | tsb_bool=>
      return    ".section .data" & newline
              &  "  ec"   & Value(ecx) & ": .long   "
              & Value(c_val_const(tv, nv)) & newline
              &  ".section .text" & newline
              &  "  movl  $ec" & Value(ecx) & ", %"
              & registre'Image(r) & newline;

    when tsb_car =>
      return    ".section .data" & newline
              &  "  ec" & Value(ecx) & ": .ascii  """
              &  Character'Val(Integer(c_val_const(tv, nv))) & """"
              & newline
              &  ".section .text" & newline
              &  "  movl  $ec" & Value(ecx) & ", %" & registre'Image(r)
              & newline;
```

```
    when tsb_nul=>
      return    ".section .data" & newline
            &  "  ec" & Value(ecx) & ": .asciz  "
            &  decls.d_tnoms.get(tn, id_str(c_val_const(tv, nv)))
            &  newline
            &  ".section .text" & newline
            &  "  movl  $ec" & Value(ecx) & ", %" & registre'Image(r)
            & newline;

    when others  => null;
  end case;
  return "";
end ga_load_address_constant;


function ga_load_address_var_local(nv: in num_var; r: in registre)
return String is
  reg: constant String:= registre'Image(r);
  desplacament: constant String:= Value(c_desp_var(tv, nv));
begin
  return "  leal " & desplacament & "(%ebp), %" & reg & newline;
end ga_load_address_var_local;

function ga_load_address_param_local(nv: in num_var; r: in registre)
return String is
  reg: constant String:= registre'Image(r);
  desplacament: constant String:= Value(c_desp_var(tv, nv));
begin
  return "  movl " & desplacament & "(%ebp), %" & reg & newline;
end ga_load_address_param_local;


function ga_load_address_var_global(nv: in num_var; r: in registre)
return String is
  desp_disp: constant Integer:= 4*Value(c_prof_proc(tp,c_np_var(tv,nv)));
  desp_display: constant String:=Value(desp_disp);
  reg: constant String:= registre'Image(r);
  desplacament: constant String:= Value(c_desp_var(tv, nv));
begin
  return  "  movl $DISP, %esi" & newline
        & "  movl " & desp_display & "(%esi), %esi" & newline
        & "  leal " & desplacament & "(%esi), %" & reg &  newline;
end ga_load_address_var_global;


function ga_load_address_param_global(nv: in num_var; r: in registre)
return String is
  desp_disp: constant Integer:= 4*Value(c_prof_proc(tp,c_np_var(tv,nv)));
  desp_display: constant String:= Value(desp_disp);
  reg: constant String:= registre'Image(r);
  desplacament: constant String:= Value(c_desp_var(tv, nv));
begin
  return  "  movl $DISP, %esi" & newline
        & "  movl " & desp_display & "(%esi), %esi" & newline
        & "  movl " & desplacament & "(%esi), %" & reg &  newline;
end ga_load_address_param_global;

--Instruccions de còpia
procedure ga_cp(i3a: in instr_3a) is
  a: constant num_var:= c_arg_nv(i3a);
  b: constant num_var:= c_arg2(i3a);
begin
  ga_escriure(ga_load(b, eax)
  & ga_store(a, eax));
end ga_cp;

-- Empram un subl a l'hora de calcular el desplaçament de l'addr
-- enlloc de l'esperat addl perque ens permet generar un codi intermitj
-- molt més elegant i B+A (A < 0 i B > 0) es equivalent a B-A (A,B >0)
procedure ga_cons_idx(i3a: in instr_3a) is
  a: constant num_var:= c_arg_nv(i3a);
  b: constant num_var:= c_arg2(i3a);
  c: constant num_var:= c_arg3(i3a);
begin
  ga_escriure(ga_load(c, eax)
            & ga_load_address(b, esi)
            & "  subl %eax, %esi" & newline
            & "  movl (%esi), %eax" & newline
            & ga_store(a, eax));
end ga_cons_idx;
```

```
-- Ídem que al cas de cons_idx
procedure ga_cp_idx(i3a: in instr_3a) is
  a: constant num_var:= c_arg_nv(i3a);
  b: constant num_var:= c_arg2(i3a);
  c: constant num_var:= c_arg3(i3a);
begin
  ga_escriure(ga_load(b, eax)
            & ga_load(c, ebx)
            & ga_load_address(a, edi)
            & "  subl %eax, %edi" & newline
            & "  movl %ebx, (%edi)" & newline);
end ga_cp_idx;


--Instruccions artimetic-logiques
procedure ga_sum(i3a: in instr_3a) is
  a: constant num_var:= c_arg_nv(i3a);
  b: constant num_var:= c_arg2(i3a);
  c: constant num_var:= c_arg3(i3a);
begin
  ga_escriure(ga_load(b, eax)
             & ga_load(c, ebx)
             & "  addl %ebx, %eax" & newline
             & ga_store(a, eax));
end ga_sum;


procedure ga_res(i3a: in instr_3a) is
  a: constant num_var:= c_arg_nv(i3a);
  b: constant num_var:= c_arg2(i3a);
  c: constant num_var:= c_arg3(i3a);
begin
  ga_escriure(ga_load(b, eax)
            & ga_load(c, ebx)
            & "  subl %ebx, %eax" & newline
            & ga_store(a, eax));
end ga_res;


procedure ga_mul(i3a: in instr_3a) is
  a: constant num_var:= c_arg_nv(i3a);
  b: constant num_var:= c_arg2(i3a);
  c: constant num_var:= c_arg3(i3a);
begin
  ga_escriure(ga_load(b, eax)
            & ga_load(c, ebx)
            & "  imul %ebx, %eax" & newline
            & ga_store(a, eax));
end ga_mul;


procedure ga_div(i3a: in instr_3a) is
  a: constant num_var:= c_arg_nv(i3a);
  b: constant num_var:= c_arg2(i3a);
  c: constant num_var:= c_arg3(i3a);
begin
  ga_escriure(ga_load(b, eax)
            & "  movl %eax, %edx" & newline
            & "  sarl $31, %edx" & newline
            & ga_load(c, ebx)
            & "  idivl  %ebx" & newline
            & ga_store(a, eax));
end ga_div;


procedure ga_modul(i3a: in instr_3a) is
  a: constant num_var:= c_arg_nv(i3a);
  b: constant num_var:= c_arg2(i3a);
  c: constant num_var:= c_arg3(i3a);
begin
  ga_escriure(ga_load(b, eax)
            & "  movl %eax, %edx" & newline
            & "  sarl $31, %edx" & newline
            & ga_load(c, ebx)
            & "  idivl  %ebx" & newline
            & ga_store(a, edx));
end ga_modul;
```

```
procedure ga_neg(i3a: in instr_3a) is
  a: constant num_var:= c_arg_nv(i3a);
  b: constant num_var:= c_arg2(i3a);
  c: constant num_var:= c_arg3(i3a);
begin
  ga_escriure(ga_load(b, eax)
            & "  negl %eax" & newline
            & ga_store(a, eax));
end ga_neg;


procedure ga_op_not(i3a: in instr_3a) is
  a: constant num_var:= c_arg_nv(i3a);
  b: constant num_var:= c_arg2(i3a);
begin
  ga_escriure(ga_load(b, eax)
            & "  notl %eax" & newline
            & ga_store(a, eax));
end ga_op_not;


procedure ga_op_and(i3a: in instr_3a) is
  a: constant num_var:= c_arg_nv(i3a);
  b: constant num_var:= c_arg2(i3a);
  c: constant num_var:= c_arg3(i3a);
begin
  ga_escriure(ga_load(b, eax)
            & ga_load(c, ebx)
            & "  andl %ebx, %eax" & newline
            & ga_store(a, eax));
end ga_op_and;


procedure ga_op_or(i3a: in instr_3a) is
  a: constant num_var:= c_arg_nv(i3a);
  b: constant num_var:= c_arg2(i3a);
  c: constant num_var:= c_arg3(i3a);
begin
  ga_escriure(ga_load(b, eax)
            & ga_load(c, ebx)
            & "  orl  %ebx, %eax" & newline
            & ga_store(a, eax));
end ga_op_or;


--Instruccions de Brancament
function ga_etiq(e: in num_etiq) return String is
  etiqueta: constant String:= Value(e);
begin
  -- Petit artifici per generar l'etiqueta main i que el gcc no es queixi
  -- L'alternativa era cercar una comanda per especificar una altra etiqueta
  -- inicial però optàrem per no fer-ho.
  -- Tot i que no ho sembli, degut al disseny del llenguatge i a com
  -- assignam les etiquetes als procediments, el primer procediment reconegut
  -- sempre serà el 'main' i tendrà etiqueta E(null_nv+1)
  -- EX:
  -- proc A is
  --   proc B is
  --     ...
  --   end proc;
  --   ...
  -- begin
  --    main bloc
  -- end proc;
  if e=null_ne+1 then
    return "main: NOP" & newline;
  else
    return "E" & etiqueta  & ": NOP" & newline;
  end if;
end ga_etiq;


function ga_goto(e: in num_etiq) return String is
  etiqueta: constant String:=Value(e);
begin
  return "  jmp  E" & etiqueta & newline;
end ga_goto;
```

```
procedure ga_etiq(i3a: in instr_3a) is
  e: constant num_etiq:= c_arg_ne(i3a);
begin
  ga_escriure(ga_etiq(e));
end ga_etiq;


procedure ga_goto(i3a: in instr_3a) is
  e: constant num_etiq:= c_arg_ne(i3a);
begin
  ga_escriure(ga_goto(e));
end ga_goto;


procedure ga_ieq_goto(i3a: in instr_3a) is
  e: constant num_etiq:= c_arg_ne(i3a);
  a: constant num_var:= c_arg2(i3a);
  b: constant num_var:= c_arg3(i3a);
  e1: num_etiq;
begin
  nova_etiq(ne, e1);

  ga_escriure(ga_load(a, eax)
            & ga_load(b, ebx)
            & "  cmpl %ebx, %eax" & newline
            & "  jne  E" & Value(e1) & newline
            & ga_goto(e)
            & ga_etiq(e1));
end ga_ieq_goto;


procedure ga_gt(i3a: in instr_3a) is
  a: constant num_var:= c_arg_nv(i3a);
  b: constant num_var:= c_arg2(i3a);
  c: constant num_var:= c_arg3(i3a);
  e, e1: num_etiq;
begin
  nova_etiq(ne, e);
  nova_etiq(ne, e1);

  ga_escriure(ga_load(b, eax)
            & ga_load(c, ebx)
            & "  cmpl %eax, %ebx" & newline
            & "  jge  E" & Value(e) & newline
            & "  movl $-1, %ecx" & newline
            & ga_goto(e1)
            & ga_etiq(e)
            & "  movl $0, %ecx" & newline
            & ga_etiq(e1)
            & ga_store(a, ecx));
end ga_gt;


procedure ga_ge(i3a: in instr_3a) is
  a: constant num_var:= c_arg_nv(i3a);
  b: constant num_var:= c_arg2(i3a);
  c: constant num_var:= c_arg3(i3a);
  e, e1: num_etiq;
begin
  nova_etiq(ne, e);
  nova_etiq(ne, e1);

  ga_escriure(ga_load(b, eax)
            & ga_load(c, ebx)
            & "  cmpl %eax, %ebx" & newline
            & "  jg  E" & Value(e) & newline
            & "  movl $-1, %ecx" & newline
            & ga_goto(e1)
            & ga_etiq(e)
            & "  movl $0, %ecx" & newline
            & ga_etiq(e1)
            & ga_store(a, ecx));
end ga_ge;
```

```
procedure ga_eq(i3a: in instr_3a) is
  a: constant num_var:= c_arg_nv(i3a);
  b: constant num_var:= c_arg2(i3a);
  c: constant num_var:= c_arg3(i3a);
  e, e1: num_etiq;
begin
  nova_etiq(ne, e);
  nova_etiq(ne, e1);
  ga_escriure(ga_load(b, eax)
            & ga_load(c, ebx)
            & "  cmpl %eax, %ebx" & newline
            & "  jne  E" & Value(e) & newline
            & "  movl $-1, %ecx" & newline
            & ga_goto(e1)
            & ga_etiq(e)
            & "  movl $0, %ecx" & newline
            & ga_etiq(e1)
            & ga_store(a, ecx));
end ga_eq;


procedure ga_neq(i3a: in instr_3a) is
  a: constant num_var:= c_arg_nv(i3a);
  b: constant num_var:= c_arg2(i3a);
  c: constant num_var:= c_arg3(i3a);
  e, e1: num_etiq;
begin
  nova_etiq(ne, e);
  nova_etiq(ne, e1);
  ga_escriure(ga_load(b, eax)
            & ga_load(c, ebx)
            & "  cmpl %eax, %ebx" & newline
            & "  je  E" & Value(e) & newline
            & "  movl $-1, %ecx" & newline
            & ga_goto(e1)
            & ga_etiq(e)
            & "  movl $0, %ecx" & newline
            & ga_etiq(e1)
            & ga_store(a, ecx));
end ga_neq;

procedure ga_lt(i3a: in instr_3a) is
  a: constant num_var:= c_arg_nv(i3a);
  b: constant num_var:= c_arg2(i3a);
  c: constant num_var:= c_arg3(i3a);
  e, e1: num_etiq;
begin
  nova_etiq(ne, e);
  nova_etiq(ne, e1);
  ga_escriure(ga_load(b, eax)
            & ga_load(c, ebx)
            & "  cmpl %eax, %ebx" & newline
            & "  jle  E" & Value(e) & newline
            & "  movl $-1, %ecx" & newline
            & ga_goto(e1)
            & ga_etiq(e)
            & "  movl $0, %ecx" & newline
            & ga_etiq(e1)
            & ga_store(a, ecx));
end ga_lt;

procedure ga_le(i3a: in instr_3a) is
  a: constant num_var:= c_arg_nv(i3a);
  b: constant num_var:= c_arg2(i3a);
  c: constant num_var:= c_arg3(i3a);
  e, e1: num_etiq;
begin
  nova_etiq(ne, e);
  nova_etiq(ne, e1);
  ga_escriure(ga_load(b, eax)
            & ga_load(c, ebx)
            & "  cmpl %eax, %ebx" & newline
            & "  jl  E" & Value(e) & newline
            & "  movl $-1, %ecx" & newline
            & ga_goto(e1)
            & ga_etiq(e)
            & "  movl $0, %ecx" & newline
            & ga_etiq(e1)
            & ga_store(a, ecx));
end ga_le;
```

```
  --Crida a procediments
  procedure ga_pmb(i3a: in instr_3a) is
    np: constant num_proc:= c_arg_np(i3a);
    desp_disp: constant Integer:= 4*Value(c_prof_proc(tp, np));
    desp_display: constant String:= Value(desp_disp);
    ocupacio_proc: constant String:= Value(c_ocup_proc(tp, np));
  begin
    empila(pproc, np);

    ga_escriure("  movl $DISP, %esi" & newline
              & "  movl " & desp_display & "(%esi), %eax" & newline
              & "  pushl %eax" & newline
              & "  pushl %ebp" & newline
              & "  movl %esp, %ebp" & newline
              & "  movl %ebp, " & desp_display & "(%esi)" & newline
              & "  subl  $" & ocupacio_proc & ", %esp" & newline);
  end ga_pmb;


  procedure ga_rtn(i3a: in instr_3a) is
    np: constant num_proc:= c_arg_np(i3a);
    desp_disp: constant Integer:= 4*Value(c_prof_proc(tp, np));
    desp_display: constant String:= Value(desp_disp);
  begin
    ga_escriure("  movl %ebp, %esp" & newline
              & "  popl %ebp" & newline
              & "  movl $DISP, %edi" & newline
              & "  popl %eax" & newline
              & "  movl %eax, " & desp_display & "(%edi)" & newline
              & "  ret" & newline);

    desempila(pproc);
  end ga_rtn;


  procedure ga_params(i3a: in instr_3a) is
    a: constant num_var:= c_arg_nv(i3a);
  begin
    ga_escriure(ga_load_address(a, eax)
              & "  pushl %eax" & newline);
  end ga_params;


  -- Empram un subl a l'hora de calcular el desplaçament de l'addr
  -- enlloc de l'esperat addl perque ens permet generar un codi intermitj
  -- molt més elegant i B+A (A < 0 i B > 0) es equivalent a B-A (A,B >0)
  procedure ga_paramc(i3a: in instr_3a) is
    a: constant num_var:=c_arg_nv(i3a);
    b: constant num_var:=c_arg2(i3a);
  begin
    ga_escriure(ga_load_address(a, eax)
              & ga_load(b, ebx)
              & "  subl %ebx, %eax" & newline
              & "  pushl %eax" & newline);
  end ga_paramc;


  procedure ga_call(i3a: in instr_3a) is
    np: constant num_proc:= c_arg_np(i3a);
    nparam: constant Integer:= 4*c_nparam_proc(tp, np);
    tamany_nparam: constant String:= Value(nparam);
  begin
    if c_tproc(tp, np)=comu then
      ga_escriure("  call E" & Value(c_etiq_proc(tp, np)) & newline
                & "  addl $" & tamany_nparam & ", %esp" & newline);
    else
      ga_escriure("  call _" & c_nom_proc(tp, np, tn) & newline
                & "  addl $" & tamany_nparam & ", %esp" & newline);
    end if;
  end ga_call;
end semantica.g_codi_ass;
```

```
with decls; use decls;
with decls.d_descripcio; use decls.d_descripcio;
with decls.d_tsimbols; use decls.d_tsimbols;

package semantica.missatges is

  procedure missatges_desc_no_es_tipus(pos: in posicio; id: in id_nom);
  procedure missatges_conflictes_declaracio(pos: in posicio; id: in id_nom);
  procedure missatges_operacio_amb_escalar(pos: in posicio);
  procedure missatges_tipus_incosistent_lit(pos: in posicio; id_tipus: in id_nom;
                                            tsb_found: in tipus_subjacent);

  procedure missatges_tipus_incosistent_id(pos: in posicio; id_expected, id_found: in id_nom);
  procedure missatges_valor_fora_rang(pos: in posicio; id_tipus: in id_nom);
  procedure missatges_rang_incorrecte(pos: in posicio);
  procedure missatges_assignacio_incorrecta(pos: in posicio);
  procedure missatges_operador_tipus(pos: in posicio; tsb_tipus: in tipus_subjacent;
                                     op: in operand);

  procedure missatges_log_operador(pos: in posicio; tsb: in tipus_subjacent);
  procedure missatges_sent_buida;
  procedure missatges_expressions_incompatibles(pos: in posicio;
                                                id_tipus1, id_tipus2: in id_nom;
                                                tsb1, tsb2: in tipus_subjacent);

  procedure missatges_no_record(pos: in posicio; id: in id_nom);
  procedure missatges_camp_no_record(pos: in posicio; id_rec, id_camp: in id_nom);
  procedure missatges_no_array(pos: in posicio; id: in id_nom);
  procedure missatges_menys_indexos_array(pos: in posicio; id_array: in id_nom);
  procedure missatges_massa_indexos_array(pos: in posicio; id_array: in id_nom);
  procedure missatges_menys_arguments_proc(pos: in posicio; id_proc: in id_nom);
  procedure missatges_massa_arguments_proc(pos: in posicio; id_proc: in id_nom);
  procedure missatges_arg_mode(pos: in posicio; id: in id_nom);
  procedure missatges_proc_mult_parentesis(pos: in posicio);
  procedure missatges_cond_bool(pos: in posicio; tsb: in tipus_subjacent);
  procedure missatges_no_definida(pos: in posicio; id: in id_nom);
  procedure missatges_no_proc(pos: in posicio);
end semantica.missatges;
```

```
with Ada.text_io; use Ada.text_IO;
with Ada.Integer_Text_IO;
with Ada.Strings.Fixed; use Ada.Strings.Fixed;

with d_queue; use d_queue;
with decls; use decls;
with decls.d_descripcio; use decls.d_descripcio;
with decls.d_tsimbols; use decls.d_tsimbols;

package body semantica.missatges is

  procedure missatges_desc_no_es_tipus(pos: in posicio; id: in id_nom) is
  begin
    put_line(pos.fila'img & ":" & pos.columna'img & ": """ & get(tn, id)
            & """ no és un tipus.");
  end missatges_desc_no_es_tipus;


  procedure missatges_conflictes_declaracio(pos: in posicio; id: in id_nom) is
  begin
    put_line(pos.fila'img & ":" & pos.columna'img & ": """ & get(tn, id)
             &""" està en conflicte amb una altre declaració.");
  end missatges_conflictes_declaracio;


  procedure missatges_operacio_amb_escalar(pos: in posicio) is
  begin
    put_line(pos.fila'img & ":" & pos.columna'img
            &": Operació il·legal=> "
            & "Aquest tipus d'assignacions sols es poden dur a terme entre escalars.");
  end missatges_operacio_amb_escalar;


  procedure missatges_tipus_incosistent_lit(pos: in posicio; id_tipus: in id_nom;
                                            tsb_found: in tipus_subjacent) is
  begin
    put_line(pos.fila'img & ":" & pos.columna'img & ": Tipus esperat """
           & get(tn,id_tipus) & """");
    put(pos.fila'img & ":" & pos.columna'img & ":");
    case tsb_found is
      when tsb_bool=>
        put_line("S'ha trobat un tipus booleà.");

      when tsb_car=>
        put_line("S'ha trobat un tipus caràcter.");

      when tsb_ent=>
        put_line("S'ha trobat un tipus enter.");

      when tsb_arr=>
        put_line("S'ha trobat un tipus array.");

      when tsb_rec=>
        put_line("S'ha trobat un tipus record.");

      when others=> null;
    end case;
  end missatges_tipus_incosistent_lit;


  procedure missatges_tipus_incosistent_id(pos: in posicio; id_expected, id_found: in id_nom)
  is
  begin
    put_line(pos.fila'img & ":" & pos.columna'img & ": Tipus esperat """
           & get(tn,id_expected) & """.");
    put_line(pos.fila'img & ":" & pos.columna'img & ": Tipus trobat """
           & get(tn,id_found) & """.");
  end missatges_tipus_incosistent_id;


  procedure missatges_valor_fora_rang(pos: in posicio; id_tipus: in id_nom) is
  begin
    put_line(pos.fila'img & ":" & pos.columna'img & ": Valor fora del rang del tipus """
             &get(tn, id_tipus) & """.");
  end missatges_valor_fora_rang;
```

```
procedure missatges_rang_incorrecte(pos: in posicio) is
begin
  put_line(pos.fila'img & ":" & pos.columna'img
          &": Rang incorrecte=> "
         & "El límit inferior ha de ser menor que el límit superior.");
end missatges_rang_incorrecte;


procedure missatges_assignacio_incorrecta(pos: in posicio) is
begin
  put_line(pos.fila'img & ":" & pos.columna'img & ": Assignació il·legal ");
end missatges_assignacio_incorrecta;


procedure missatges_operador_tipus(pos: in posicio; tsb_tipus: in tipus_subjacent;
                                   op: in operand) is
begin
  case op is
    when major | majorigual | igual | diferent | menorigual | menor=>
      put(pos.fila'img & ":" & pos.columna'img
          & ": Els operadors relacionals no han estat definits ");

    when sum=>
      put(pos.fila'img & ":" & pos.columna'img & ": L'operador ""+"" no està definit ");

    when res=>
      put(pos.fila'img & ":" & pos.columna'img & ": L'operador ""-"" no està definit ");

    when prod=>
      put(pos.fila'img & ":" & pos.columna'img & ": L'operador ""*"" no està definit ");

    when quoci=>
      put(pos.fila'img & ":" & pos.columna'img & ": L'operador ""/"" no està definit ");

    when modul=>
      put(pos.fila'img & ":" & pos.columna'img & ": L'operador ""mod"" no està definit ");

    when neg_log=>
      put(pos.fila'img & ":" & pos.columna'img & ": L'operador ""not"" no està definit ");

    when neg_alg=>
      put(pos.fila'img & ":" & pos.columna'img
        & ": L'operador ""- unari"" no esta definit ");

    when others=> null;
  end case;

  case tsb_tipus is
    when tsb_bool=>
      put_line("per a un tipus booleà.");

    when tsb_car=>
      put_line("per a un tipus caràcter.");

    when tsb_ent=>
      put_line("per a un tipus enter.");

    when tsb_arr=>
      put_line("per a un tipus array.");

    when tsb_rec=>
      put_line("per un tipus record.");

    when others=> null;
  end case;
end missatges_operador_tipus;
```

```
procedure missatges_log_operador(pos: in posicio; tsb: in tipus_subjacent) is
begin
  put_line(pos.fila'img & ":" & pos.columna'img
           & ": Els operadors llogics or/and no son definits ");
  case tsb is
    when tsb_bool=>
      put_line("per un tipus boolea.");

    when tsb_car=>
      put_line("per un tipus caracter.");

    when tsb_ent=>
      put_line("per un tipus enter.");

    when tsb_arr=>
      put_line("per un tipus array.");

    when tsb_rec=>
      put_line("per un tipus record.");

    when others=> null;
  end case;
end missatges_log_operador;

procedure missatges_expressions_incompatibles(pos: in posicio;
                                              id_tipus1, id_tipus2: in id_nom;
                                              tsb1, tsb2: in tipus_subjacent) is
begin
  put(pos.fila'img & ":" & pos.columna'img & ": El tipus """ );
  if id_tipus1 /= null_id then
    put(get(tn, id_tipus1));
  else
    case tsb1 is
      when tsb_bool=>
        put("boolea");

      when tsb_car=>
        put("caracter");

      when tsb_ent=>
        put("enter");

      when tsb_arr=>
        put("array");

      when tsb_rec=>
        put("record");
      when others=> null;
    end case;
  end if;
  put(""" no es compatible amb el tipus """);
  if id_tipus2 /= null_id then
    put_line(get(tn, id_tipus2) & """.");
  else
    case tsb2 is
      when tsb_bool=>
        put_line("boolea"".");

      when tsb_car=>
        put_line("caracter"".");

      when tsb_ent=>
        put_line("enter"".");

      when tsb_arr=>
        put_line("array"".");

      when tsb_rec=>
        put_line("record"".");
      when others=> null;
    end case;
  end if;
end missatges_expressions_incompatibles;

procedure missatges_sent_buida is
begin
  put("Sentencia esperada, no s'admeten blocs de senténcies buides, "
    & "si voleu deixar-los buits empreu la keyword ""null"".");
end missatges_sent_buida;
```

```
procedure missatges_menys_indexos_array(pos: in posicio; id_array: in id_nom) is
begin
  put_line(pos.fila'img & ":" & pos.columna'img
         & ": Nombre insuficient d'indexos per a l'array """ & get(tn, id_array) & """.");
end missatges_menys_indexos_array;


procedure missatges_massa_indexos_array(pos: in posicio; id_array: in id_nom) is
begin
  put_line(pos.fila'img & ":" & pos.columna'img
          & ": Nombre major d'indexos que els acceptats al rang de l'array """
          & get(tn, id_array) & """.");
end missatges_massa_indexos_array;


procedure missatges_no_record(pos: in posicio; id: in id_nom) is
begin
  put_line(pos.fila'img & ":" & pos.columna'img
          & ": El tipus """ & get(tn, id) & """ no es un record.");
end missatges_no_record;


procedure missatges_camp_no_record(pos: in posicio; id_rec, id_camp: in id_nom) is
begin
  put_line(pos.fila'img & ":" & pos.columna'img
         & ": El record """ & get(tn, id_rec)
         & """ no té cap camp amb el nom """ & get(tn, id_camp) & """.");
end missatges_camp_no_record;

procedure missatges_no_array(pos: in posicio; id: in id_nom) is
begin
  put_line(pos.fila'img & ":" & pos.columna'img
        & ": El tipus """ & get(tn, id) & """ no és un array.");
end missatges_no_array;

procedure missatges_menys_arguments_proc(pos: in posicio; id_proc: in id_nom) is
begin
  put_line(pos.fila'img & ":" & pos.columna'img
         & ": Nombre insuficient d'arguments pel al procediment """
         & get(tn, id_proc) & """.");
end missatges_menys_arguments_proc;


procedure missatges_massa_arguments_proc(pos: in posicio; id_proc: in id_nom) is
begin
  put_line(pos.fila'img & ":" & pos.columna'img
        & ": Nombre major d'arguments que els acceptats pel procediment """
        & get(tn, id_proc) & """.");
end missatges_massa_arguments_proc;


procedure missatges_arg_mode(pos: in posicio; id: in id_nom) is
begin
  put_line(pos.fila'img & ":" & pos.columna'img
         & ": L'argument """ & get(tn,id) & " no compleix amb el seu mode");
end missatges_arg_mode;


procedure missatges_proc_mult_parentesis(pos: in posicio) is
begin
  put_line(pos.fila'img & ":" & pos.columna'img
         & ": Paréntesis no lligats a cap expressió.");
end missatges_proc_mult_parentesis;
```

```
  procedure missatges_cond_bool(pos: in posicio; tsb: in tipus_subjacent) is
  begin
    put_line(pos.fila'img & ":" & pos.columna'img & ": Tipus esperat ""Boolean""");
    put(pos.fila'img & ":" & pos.columna'img & ":");
    case tsb is
      when tsb_bool=>
        put_line("S'ha trobat un tipus booleà.");

      when tsb_car=>
        put_line("S'ha trobat un tipus caràcter.");

      when tsb_ent=>
        put_line("S'ha trobat un tipus enter.");

      when tsb_arr=>
        put_line("S'ha trobat un tipus array.");

      when tsb_rec=>
        put_line("S'ha trobat un tipus record.");

      when others=> null;
    end case;
  end missatges_cond_bool;


  procedure missatges_no_definida(pos: in posicio; id: in id_nom) is
  begin
    put_line(pos.fila'img & ":" & pos.columna'img
             &": La entitat """ & get(tn, id) & """ no està definida.");
  end missatges_no_definida;


  procedure missatges_no_proc(pos: in posicio) is
  begin
    put_line(pos.fila'img & ":" & pos.columna'img
           & ": S'espera un procediment, no un array.");
  end missatges_no_proc;

end semantica.missatges;
```