```ada
package decls is

  pragma pure;

  type valor is new integer;
  type despl is new integer;

  -- 4Bytes x enter
  ocup_ent: constant despl:= 4;
  -- 4Bytes x char (simplificacio)
  ocup_char: constant despl:= ocup_ent;
  -- 4Bytes x bool (simplificacio)
  ocup_bool: constant despl:= ocup_char;
  -- Aquests 2 serien els que teoricament hauriem d'emprar però per
  -- simplicitat no ho feim
  -- 1Byte x char
  ocup_char_compressed: constant despl:= 1;
  -- 1Byte x boolean
  ocup_bool_compressed: constant despl:= ocup_char_compressed;

  type tidx is (
    positiu,
    negatiu
  );


  max_id: constant integer:=997;
  max_str: constant integer:=499;

  type id_nom is new natural range 0..max_id;
  type id_str is new natural range 0..max_str;

  null_id: constant id_nom:= id_nom'First;


  max_var: constant integer:= 1023;
  max_proc: constant integer:= 511;
  max_etiq: constant integer:= 1023;

  type num_var is new natural range 0..max_var;
  type num_proc is new natural range 0..max_proc;
  type num_etiq is new natural range 0..max_etiq;

  null_nv: constant num_var:= num_var'First;
  null_np: constant num_proc:= num_proc'First;
  null_ne: constant num_etiq:= num_etiq'First;

end decls;
```

```
with decls.d_descripcio;
package decls.d_arbre is
  type node;
  type pnode is access node;
  subtype atribut is pnode;
  type tnode is (
    nd_null,
    nd_root,
    nd_proc,
    nd_decls,
    nd_decl,
    nd_decl_var,
    nd_decl_const,
    nd_decl_t,
    nd_decl_t_cont_type,
    nd_decl_t_cont_record,
    nd_decl_t_cont_arry,
    nd_dcamps,
    nd_dcamp,
    nd_rang,
    nd_c_proc,
    nd_args,
    nd_arg,
    nd_mode,
    nd_lid,
    nd_idx,
    nd_idx_cont,
    nd_sents,
    nd_sents_nob,
    nd_sent,
    nd_siter,
    nd_scond,
    nd_scrida,
    nd_sassign,
    nd_ref,
    nd_iproc,
    nd_var,
    nd_qs,
    nd_q,
    nd_arry,
    nd_rec,
    nd_lexpr_arry,
    nd_expr,
    nd_and,
    nd_or,
    nd_eop,
    nd_et,
    nd_lexpr,
    nd_id,
    nd_lit,
    nd_op_rel
  );

  type tmode is (
    md_in,
    md_in_out
  );
  type posicio is
    record
      fila: natural;
      columna: natural;
    end record;

  type operand is (
    nul,
    menor,
    major,
    menorigual,
    majorigual,
    igual,
    diferent,
    sum,
    res,
    prod,
    quoci,
    pot,
    modul,
    neg_log,
    neg_alg
  );
```

```
type node(tn: tnode:= nd_null) is
  record
    case tn is
      when nd_null =>
        null;

      when nd_root =>
        p: pnode;

      when nd_id =>
        id_id: id_nom;
        id_pos: posicio;

      when nd_lit =>
        lit_val: valor;
        lit_pos: posicio;
        lit_tipus: decls.d_descripcio.tipus_subjacent;

      when nd_op_rel =>
        orel_tipus: operand;

      when nd_var =>
        var_nv: num_var;
        var_ocup: despl;

      when nd_lid =>
        lid_seg: pnode;
        lid_id: pnode;

      when nd_mode =>
        mode_tipus: tmode;

      when nd_c_proc =>
        cproc_id: pnode;
        cproc_np: num_proc;
        cproc_args: pnode;

      when nd_proc =>
        proc_cproc: pnode;
        proc_decls: pnode;
        proc_sents: pnode;

      when nd_iproc =>
        iproc_np: num_proc;

      when nd_args =>
        args_args: pnode;
        args_arg: pnode;

      when nd_arg =>
        arg_tipus: pnode;
        arg_lid: pnode;
        arg_mode: tmode;

      when nd_decls =>
        decls_decls: pnode;
        decls_decl: pnode;

      when nd_decl =>
        decl_real: pnode;

      when nd_dcamps =>
        dcamps_dcamps: pnode;
        dcamps_dcamp: pnode;

      when nd_dcamp =>
        dcamp_decl: pnode;

      when nd_decl_var =>
        dvar_lid: pnode;
        dvar_tipus: pnode;

      when nd_decl_const =>
        dconst_lid: pnode;
        dconst_tipus: pnode;
        dconst_valor: pnode;

      when nd_decl_t =>
        dt_id: pnode;
        dt_cont: pnode;
```

```
when nd_decl_t_cont_type =>
  dtcont_rang: pnode;

when nd_decl_t_cont_record =>
  dtcont_camps: pnode;

when nd_decl_t_cont_arry =>
  dtcont_idx: pnode;
  dtcont_tipus: pnode;

when nd_rang =>
  rang_id: id_nom;
  rang_linf: pnode;
  rang_lsup: pnode;

when nd_idx =>
  idx_tipus: tidx;
  idx_cont: pnode;

when nd_idx_cont =>
  idxc_valor: pnode;

when nd_sents  =>
  sents_cont: pnode;

when nd_sents_nob  =>
  snb_snb: pnode;
  snb_sent: pnode;

when nd_sent =>
  sent_sent: pnode;

when nd_siter =>
  siter_expr: pnode;
  siter_sents: pnode;

when nd_scond =>
  scond_expr: pnode;
  scond_sents: pnode;
  scond_esents: pnode;

when nd_scrida =>
  scrida_ref: pnode;

when nd_sassign =>
  sassign_ref: pnode;
  sassign_expr: pnode;

when nd_ref =>
  ref_id: pnode;
  ref_qs: pnode;

when nd_qs =>
  qs_qs: pnode;
  qs_q: pnode;

when nd_q =>
  q_contingut: pnode;

when nd_arry =>
  arry_lexpr: pnode;
  -- constants de despl calc per el compilador
  arry_tb: num_var;
  arry_tw: num_var;

when nd_lexpr_arry =>
  lexpra_cont: pnode;
  lexpra_expr: pnode;
  -- constant del compilador (lsup-linf+1)
  lexpra_tu: num_var;

when nd_rec =>
  -- despl constant del camp.
  rec_td: num_var;

when nd_expr =>
  expr_e: pnode;
```

```
        when nd_and | nd_or =>
          e_ope: pnode;
          e_opd: pnode;

        when nd_eop =>
          eop_ope: pnode;
          eop_opd: pnode;
          eop_operand: operand;

        when nd_et =>
          et_cont: pnode;

        when nd_lexpr =>
          lexpr_cont: pnode;
          lexpr_expr: pnode;

      end case;
    end record;

end decls.d_arbre;
```

```
package decls.d_descripcio is
  type tipus_subjacent is (
    tsb_bool,
    tsb_car,
    tsb_ent,
    tsb_arr,
    tsb_rec,
    tsb_nul
  );

  type descr_tipus(tsb: tipus_subjacent:= tsb_nul) is
    record
      ocup: despl;
      case tsb is
        when tsb_bool | tsb_car | tsb_ent  =>
          linf,lsup: valor;
        when tsb_arr                       =>
          tcomp: id_nom;
          b: valor;
        when tsb_rec | tsb_nul             =>
          null;
        end case;
      end record;

    type tipus_descr is (
      dnula,
      dvar,
      dconst,
      dindx,
      dtipus,
      dcamp,
      dproc,
      dargc
    );

    type descripcio(td: tipus_descr:= dnula) is
      record
        case td is
          when dnula  =>
            null;
          when dvar   =>
            tv: id_nom; -- tipus de la variable
            nv: num_var;
          when dconst =>
            tc: id_nom; -- tipus de la constant
            vc: valor; -- valor de la constant
          when dindx  =>
            tind: id_nom;
          when dtipus =>
            dt: descr_tipus;
          when dcamp  =>
            tcmp: id_nom; -- tipus del camp
            dcmp: despl;
          when dproc  =>
            np: num_proc;
          when dargc  =>
            ta: id_nom; -- tipus de l'argument
            na: num_var;
        end case;
      end record;
end decls.d_descripcio;
```

```
with Ada.Sequential_IO;
with decls.d_descripcio; use decls.d_descripcio;
with decls.d_tsimbols; use decls.d_tsimbols;
with decls.d_tnoms; use decls.d_tnoms;
package decls.d_c3a is
  type discr_instruccio is (comu, proc, etiq);
  type instr_3a(d: discr_instruccio:= comu) is private;
  type instr_3a_bin is private;
  type tinstruccio is (
    cp,
    cp_idx,
    cons_idx,
    sum,
    res,
    mul,
    div,
    modul,
    neg,
    op_not,
    op_and,
    op_or,
    etiq,
    go_to,
    ieq_goto,
    gt,
    ge,
    eq,
    neq,
    le,
    lt,
    pmb,
    rtn,
    call,
    params,
    paramc
  );
  type tproc is (std, comu);

  type tvariables is limited private;
  type tprocediments is limited private;

  procedure nova_var(nv: in out num_var; tv: in out tvariables;
                     tp: in out tprocediments; np: in num_proc;
                     ocup: in despl; t: out num_var);
  procedure nova_var_const(nv: in out num_var; tv: in out tvariables; val: in valor;
                           tsb: in tipus_subjacent; t: out num_var);
  procedure nou_arg(nv: in out num_var; tv: in out tvariables;
                    tp: in out tprocediments; np: in num_proc;
                    offset: in despl; t: out num_var);

  procedure nou_proc(np: in out num_proc; tp: in out tprocediments; e: in num_etiq;
                     prof: in profunditat; nparam: in natural; p: out num_proc);
  procedure nou_proc_std(np: in out num_proc; tp: in out tprocediments;
                         id: in id_nom; prof: in profunditat;
                         nparam: in natural; p: out num_proc);
  procedure nova_etiq (ne: in out num_etiq; e: out num_etiq);


  -- Funcions de consulta i conversió.Omeses per conveniència
  function Value(*) return instr_3a;
  function Imatge(i3a: in instr_3a; tv: in tvariables; tp: in tprocediments) return String;


  -- Conversió d'instrucció normal a instrucció binària (per guardar al
  -- fitxer)
  function To_i3a_bin(i3a: in instr_3a) return instr_3a_bin;
  function To_i3a(i3a_b: in instr_3a_bin) return instr_3a;

  -- Funcions per consultar els camps de les variables, instruccions, procediments
  -- omeses per conveniència
  function consulta_*(*) return *;

  --Procediment per actualitzar els camps desp i ocup_vl de les variables/procs
  procedure calcul_desplacaments (tv: in out tvariables; nv: in num_var;
                                  tp: in out tprocediments; np: in num_proc);

  -- Procediment per actualitzar el nombre d'args d'un procediment
  procedure act_proc_args(tp: in out tprocediments; np: in num_proc; nargs: in natural);
```

```
private

  type instr_3a (d: discr_instruccio:= comu) is
    record
      t: tinstruccio;
      b: num_var;
      c: num_var;
      case d is
        when comu =>
          nv: num_var;
        when proc =>
          np: num_proc;
        when etiq =>
          ne: num_etiq;
      end case;
    end record;

  -- Aquest es el tipus d'instruccions escrites al fitxer binári ja que simplifiquen
  -- molt la seua gestió
  type instr_3a_bin is
    record
      t: tinstruccio;
      a: integer;
      b: integer;
      c: integer;
    end record;

  type tvar is (esvar, esconst);
  type e_tvar (tv: tvar) is
    record
      case tv is
        when esvar =>
          np: num_proc;
          ocup: despl;
          desp: despl;
        when esconst =>
          val: valor;
          tsb: tipus_subjacent;
      end case;
    end record;

  type pe_tvar is access e_tvar;


  type e_tproc (tp: tproc)is
    record
      prof: profunditat;
      nparam: natural;
      case tp is
        when comu =>
          e: num_etiq;
          ocup_vl: despl;
        when std =>
          id: id_nom;
      end case;
    end record;

  type pe_tproc is access e_tproc;

  type tvariables is array (num_var) of pe_tvar;
  type tprocediments is array (num_proc) of pe_tproc;

end decls.d_c3a;
```

```
with decls.d_tnoms;
with semantica; use semantica;
package body decls.d_c3a is

  procedure nova_var(nv: in out num_var; tv: in out tvariables;
                     tp: in out tprocediments; np: in num_proc;
                     ocup: in despl; t: out num_var) is
  begin
    nv:= nv+1;
    tv(nv):= new e_tvar'(esvar, np, ocup, 0);
    t:= nv;
  end nova_var;


  procedure nova_var_const(nv: in out num_var; tv: in out tvariables; val: in valor;
                          tsb: in tipus_subjacent; t: out num_var) is
  begin
    nv:= nv+1;
    tv(nv):= new e_tvar'(esconst, val, tsb);
    t:= nv;
  end nova_var_const;

  procedure nou_arg(nv: in out num_var; tv: in out tvariables;
                    tp: in out tprocediments; np: in num_proc;
                    offset: in despl; t: out num_var) is
  begin
    nv:= nv+1;
    tv(nv):= new e_tvar'(esvar, np, ocup_ent, offset);
    t:= nv;
  end nou_arg;


  procedure nou_proc(np: in out num_proc; tp: in out tprocediments; e: in num_etiq;
                     prof: in profunditat; nparam: in natural; p: out num_proc) is
  begin
    np:= np+1;
    tp(np):= new e_tproc'(comu, e=> e, prof=> prof, ocup_vl=> 0,
                          nparam=> nparam);
    p:= np;
  end nou_proc;


  procedure nou_proc_std(np: in out num_proc; tp: in out tprocediments;
                         id: in id_nom; prof: in profunditat;
                         nparam: in natural; p: out num_proc) is
  begin
    np:= np+1;
    tp(np):= new e_tproc'(std, id=> id, prof=> prof, nparam=> nparam);
    p:= np;
  end nou_proc_std;


  procedure nova_etiq (ne: in out num_etiq; e: out num_etiq) is
  begin
    ne:= ne+1;
    e:= ne;
  end nova_etiq;

  -- totes les funcions Value s'han omés per conveniència
  function Value(t: in tinstruccio; *) return instr_3a is
  begin
    return (*);
  end Value;


  function Imatge(i3a: in instr_3a; tv: in tvariables; tp: in tprocediments) return String is
  begin
    -- omés per conveniència
  end Imatge;


  function To_i3a_bin(i3a: in instr_3a) return instr_3a_bin is
  begin
    -- omés per conveniència
  end To_i3a_bin;
  function To_i3a(i3a_b: in instr_3a_bin) return instr_3a is
  begin
    -- omés per conveniència
  end To_i3a;


  -- totes les funcions de consulta s'han omés per conveniéncia
  function consulta_*(i3a: in instr_3a;*) return * is
  begin
    return i3a.*;
  end consulta_tipus;


  procedure calcul_desplacaments (tv: in out tvariables; nv: in num_var;
                                  tp: in out tprocediments; np: in num_proc) is
    desp: despl;
    var: e_tvar(esvar);
    ldesp: array(num_proc range null_np+1..np) of despl;
  begin
    for ip in null_np+1..np loop
      ldesp(ip):= 0;
    end loop;

    for iv in null_nv+1..nv loop
      if tv(iv).all.tv = esvar  and then tv(iv).all.desp <= 0 then
        var:= tv(iv).all;
        if tp(var.np).all.tp = comu then
          desp:= ldesp(var.np);
          if desp = 0 then
            ldesp(var.np):= ocup_ent;
            desp:= ocup_ent;
          end if;
          tv(iv).all.desp:= -desp;
          ldesp(var.np):= desp + var.ocup;
        end if;
      end if;
    end loop;

    for ip in null_np+1..np loop
      if tp(ip).all.tp = comu then
        tp(ip).all.ocup_vl:= ldesp(ip);
      end if;
    end loop;
  end calcul_desplacaments;


  procedure act_proc_args(tp: in out tprocediments; np: in num_proc; nargs: in natural) is
  begin
    tp(np).nparam:= nargs;
  end act_proc_args;

end decls.d_c3a;
```

```
with Ada.Containers; use Ada.Containers;
package decls.d_tnoms is

    type tnoms is limited private;

   --Noms
    procedure empty(tn: out tnoms);
    procedure put(tn: in out tnoms; nom: in String; ident: out id_nom);
    function get(tn: in tnoms; ident: in id_nom)return string;

   --Strings/Literals
    procedure put(tn: in out tnoms; text: in string; ids: out id_str);
    function get(tn: in tnoms; ids: in id_str) return string;

   --Excepcions
    space_overflow,bad_use: exception;

private
    max_ch: constant Natural:= (max_id+max_str)*64;
    maxid: constant id_nom:= id_nom(max_id);
    maxstr: constant id_str:= id_str(max_str);
    b: constant Ada.Containers.Hash_Type:= Ada.Containers.Hash_Type(max_id);

    subtype hash_index is Ada.Containers.Hash_Type range 0..b-1;
    type list_item is
        record
            psh:id_nom;
            ptc:natural;
        end record;
    type id_table is array (id_nom) of list_item;
    type str_table is array (id_str) of Natural;
    type disp_table is array (hash_index) of id_nom;
    subtype char_table is String(1..max_ch);

    type tnoms is
        record
            td: disp_table:= (others=> null_id);
            tid: id_table;
            ts: str_table;
            tc: char_table;
            nid: id_nom:= null_id;--num idents
            ns: id_str:= 0;--num strings
            nc: Natural:= 0;--num chars idents
            ncs: Natural:= max_ch;--num chars strings
        end record;
end decls.d_tnoms;
```

```ada
with Ada.Strings.Hash; use ada.Strings;
with Ada.Characters.Handling; use Ada.Characters.Handling;
with semantica.missatges; use semantica.missatges;
package body decls.d_tnoms is
  --Auxiliar operations:
  procedure save_name(tc: in out char_table; nom: in string;
                      nc: in out integer) is
  begin
    for i in nom'Range loop
      nc:= nc+1; tc(nc):= nom(i);
    end loop;
  end save_name;


  procedure save_string(tc: in out char_table; text: in string;
                        ncs: in out integer) is
  begin
    for i in reverse text'Range loop
      ncs:= ncs-1; tc(ncs):= text(i);
    end loop;
  end save_string;


  function equal(nom: in string; tn: in tnoms; p: in id_nom)
  return boolean is
    tid: id_table renames tn.tid;
    tc: char_table renames tn.tc;
    nid: id_nom renames tn.nid;
    pi,pf: natural;
    i,j:natural;
  begin
    pi:= tid(p-1).ptc+1; pf:= tid(p).ptc;
    i:= nom'first;j:= pi;
    while nom(i)=tc(j) and i<nom'Last and j<pf loop
      i:= i+1; j:= j+1;
    end loop;
    return nom(i)=tc(j) and i=nom'Last and j=pf;
  end equal;

  -- ******************************************************************
  procedure empty(tn: out tnoms)is
    td: disp_table renames tn.td;
    tid: id_table renames tn.tid;
    ts: str_table renames tn.ts;
    nid: id_nom renames tn.nid;
    ns: id_str renames tn.ns;
    nc: integer renames tn.nc;
    ncs: integer renames tn.ncs;
  begin
    for i in hash_index loop td(i):=null_id; end loop;
    nid:= null_id; ns:= 0; nc:=0; ncs:= max_ch;
    tid(null_id):= (null_id, nc); ts(0):= max_ch;
  end empty;


  procedure put(tn: in out tnoms; nom: in string; ident: out id_nom) is
    td: disp_table renames tn.td;
    tid: id_table renames tn.tid;
    tc: char_table renames tn.tc;
    nid: id_nom renames tn.nid;
    nc: integer renames tn.nc;
    ncs: integer renames tn.ncs;
    i: hash_type;
    p: id_nom;
    nm: String:= To_Lower(nom);
  begin
    i:= hash(nm) mod b; p:= td(i);
    while p/=null_id and then not equal(nm,tn,p) loop
      p:= tid(p).psh;
    end loop;
    if p=null_id then
      if nid=maxid then raise space_overflow; end if;
      if nc+nom'Length>ncs then raise space_overflow; end if;
      save_name(tc, nm, nc);
      nid:= nid+1; tid(nid):= (td(i),nc);
      td(i):=nid; p:=nid;
    end if;
    ident:= p;
  end put;
  function get(tn: in tnoms; ident: in id_nom) return string is
    tid: id_table renames tn.tid;
    tc: char_table renames tn.tc;
    nid: id_nom renames tn.nid;
    i,j: integer;
  begin
    if ident=null_id or ident>nid then raise bad_use; end if;
    i:= tid(ident-1).ptc+1; j:= tid(ident).ptc;
    return tc(i..j);
  end get;


  procedure put(tn: in out tnoms; text: in string;ids: out id_str) is
    tc: char_table renames tn.tc;
    ts: str_table renames tn.ts;
    ns: id_str renames tn.ns;
    ncs: integer renames tn.ncs;
    nc: integer renames tn.nc;
  begin
    if ns=maxstr then raise space_overflow; end if;
    if ncs-text'Length<nc then raise space_overflow; end if;
    save_string(tc, text, ncs);
    ns:= ns+1; ts(ns):= ncs;
    ids:=ns;
  end put;


  function get(tn: in tnoms; ids: in id_str) return string is
    tc: char_table renames tn.tc;
    ts: str_table renames tn.ts;
    ns: id_str renames tn.ns;
    i,j: integer;
  begin
    if ids=0 or ids>ns then raise bad_use; end if;
    j:= ts(ids-1)-1; i:= ts(ids);
    return tc(i..j);
  end get;

end decls.d_tnoms;
```

```ada
with decls.d_descripcio; use decls.d_descripcio;
package decls.d_tsimbols is
    type tsimbols is limited private;

    type iterador_index is private;
    type iterador_arg is private;

    type profunditat is private;
    -- Operacions generals
    procedure empty(ts: out tsimbols);
    procedure put(ts: in out tsimbols; id: in id_nom; d: in descripcio; error: out boolean);
    function get(ts: in tsimbols; id: in id_nom) return descripcio;
    procedure update(ts: in out tsimbols; id: in id_nom; d: in descripcio);
    -- Operacions de record
    procedure put_camp(ts: in out tsimbols; idr,idc: in id_nom;
                       dc: in descripcio; error: out boolean);
    function get_camp(ts: in tsimbols; idr,idc: in id_nom) return descripcio;
    -- Operacions d'array
    procedure put_index(ts: in out tsimbols; ida: in id_nom; di: in descripcio);
    procedure first(ts: in tsimbols; ida: in id_nom; it: out iterador_index);
    procedure next(ts: in tsimbols; it: in out iterador_index);
    function get(ts: in tsimbols; it: in iterador_index) return descripcio;
    function is_valid(it: in iterador_index) return boolean;
    -- Operacions de procediment
    procedure put_arg(ts: in out tsimbols; idp,ida: in id_nom;
                      da: in descripcio; error: out boolean);
    procedure first(ts: in tsimbols; idp: in id_nom; it: out iterador_arg);
    procedure next(ts: in tsimbols; it: in out iterador_arg);
    procedure get(ts: in tsimbols; it: in iterador_arg; ida: out id_nom; da: out descripcio);
    function is_valid(it: in iterador_arg) return boolean;
    -- Operacions del compilador!
    procedure enter_block(ts: in out tsimbols);
    procedure exit_block(ts: in out tsimbols);

    function get_prof(ts: in tsimbols) return profunditat;
    function "<"(prof1, prof2: in profunditat) return boolean;
    function value(prof: in profunditat) return integer;
    no_es_tipus, no_es_record, no_es_array, no_es_proc, mal_us: exception;
private
    type index_expansio is new integer range 0..max_id;
    type profunditat is new integer range -1..100;

    type te_item;
    type td_item is
        record
            prof: profunditat:= 0;
            d: descripcio:= (td => dnula);
            next: index_expansio:= 0;
        end record;

    type te_item is
        record
            id: id_nom;
            prof: profunditat;
            d: descripcio;
            next: index_expansio;
        end record;

    type tdescripcio is array (id_nom) of td_item;
    type texpansio is array (index_expansio) of te_item;
    type tblocks is array (profunditat) of index_expansio;

    type tsimbols is
        record
            prof: profunditat:=1;
            td: tdescripcio:=(others=>(0,(td => dnula),0));
            te: texpansio;
            tb: tblocks:=(others=>0);
        end record;

    type iterador_index is
        record
            ie: index_expansio;
        end record;

    type iterador_arg is
        record
            ie: index_expansio;
        end record;
end decls.d_tsimbols;
```

```
with semantica; use semantica;
with semantica.missatges; use semantica.missatges;
package body decls.d_tsimbols is

  procedure empty(ts: out tsimbols) is
    td: tdescripcio renames ts.td;
    tb: tblocks renames ts.tb;
    prof: profunditat renames ts.prof;
  begin
    for id in id_nom loop
      td(id):= (0,(td => dnula),0);
    end loop;
    prof:= 0; tb(prof):= 0;
    prof:= 1; tb(prof):= tb(prof-1);
  end empty;


  procedure put(ts: in out tsimbols; id: in id_nom; d: in descripcio; error: out boolean) is
    td: tdescripcio renames ts.td;
    te: texpansio renames ts.te;
    tb: tblocks renames ts.tb;
    prof: profunditat renames ts.prof;
    ie: index_expansio;
  begin
    error:= false;
    if td(id).prof=prof then
      error:= true;
    end if;
    if not error then
      ie:= tb(prof); ie:= ie+1; tb(prof):= ie;
      te(ie).prof:= td(id).prof; te(ie).d:= td(id).d;
      td(id).prof:= prof;
      td(id).d:= d;
      te(ie).id:= id; te(ie).next:= 0;
    end if;
  end put;


  function get(ts: in tsimbols; id: in id_nom) return descripcio is
    td: tdescripcio renames ts.td;
  begin
    return td(id).d;
  end get;


  procedure put_camp(ts: in out tsimbols; idr,idc: in id_nom;
                     dc: in descripcio; error: out boolean) is
    td: tdescripcio renames ts.td;
    te: texpansio renames ts.te;
    tb: tblocks renames ts.tb;
    prof: profunditat renames ts.prof;
    ie: index_expansio;
  begin
    if td(idr).d.td /= dtipus then raise no_es_tipus; end if;
    if td(idr).d.dt.tsb /= tsb_rec then
      raise no_es_record;
    end if;
    error:=false;
    ie:= td(idr).next;
    while ie /= 0 and then te(ie).id /= idc loop
      ie:= te(ie).next;
    end loop;
    if ie /= 0 then error:= true; end if;
    if not error then
      ie:= tb(prof); ie:= ie+1; tb(prof):= ie;
      te(ie).id:= idc; te(ie).prof:= -1; te(ie).d:= dc;
      te(ie).next:= td(idr).next; td(idr).next:= ie;
    end if;
  end put_camp;
```

```
function get_camp(ts: in tsimbols; idr,idc: in id_nom)
return descripcio is
  td: tdescripcio renames ts.td;
  te: texpansio renames ts.te;
  ie: index_expansio;
  d: descripcio;
begin
  if td(idr).d.td /= dtipus then raise no_es_tipus; end if;
  if td(idr).d.dt.tsb /= tsb_rec then raise no_es_record; end if;
  ie:= td(idr).next;
  while ie /= 0 and then te(ie).id /= idc loop
    ie:= te(ie).next;
  end loop;
  if ie=0 then d:= (td => dnula);
  else d:= te(ie).d;
  end if;
  return d;
end get_camp;

procedure update(ts: in out tsimbols; id: in id_nom; d: in descripcio)
is
  td: tdescripcio renames ts.td;
begin
  td(id).d:=d;
end update;


procedure put_index(ts: in out tsimbols; ida: in id_nom; di: in descripcio)
is
  td: tdescripcio renames ts.td;
  tb: tblocks renames ts.tb;
  te: texpansio renames ts.te;
  prof: profunditat renames ts.prof;
  ie, iep: index_expansio;
begin
  if td(ida).d.td /= dtipus then raise no_es_tipus; end if;
  if td(ida).d.dt.tsb /= tsb_arr then raise no_es_array; end if;
  iep:=0; ie:= td(ida).next;
  while ie /= 0 loop
    iep:= ie; ie:= te(ie).next;
  end loop;
  ie:= tb(prof); ie:= ie+1; tb(prof):= ie;
  te(ie).id:= null_id; te(ie).d:= di; te(ie).prof:= -1;
  if iep = 0 then td(ida).next:= ie;
  else te(iep).next:= ie;
  end if;
    te(ie).next:= 0;
end put_index;


procedure first(ts: in tsimbols; ida: in id_nom; it: out iterador_index)
is
  td: tdescripcio renames ts.td;
begin
  if td(ida).d.td /= dtipus then raise no_es_tipus; end if;
  if td(ida).d.dt.tsb /= tsb_arr then raise no_es_array; end if;
  it.ie:= td(ida).next;
end first;


procedure next(ts: in tsimbols; it: in out iterador_index) is
  te: texpansio renames ts.te;
begin
  if it.ie=0 then raise mal_us; end if;
  it.ie:= te(it.ie).next;
end next;


function get(ts: in tsimbols; it: in iterador_index) return descripcio
is
  te: texpansio renames ts.te;
begin
  if it.ie=0 then raise mal_us; end if;
  return te(it.ie).d;
end get;

function is_valid(it: in iterador_index) return boolean is
begin
  return it.ie /= 0;
end is_valid;
```

```
procedure put_arg(ts: in out tsimbols; idp,ida: in id_nom;
                  da: in descripcio;error: out boolean) is
  td: tdescripcio renames ts.td;
  te: texpansio renames ts.te;
  tb: tblocks renames ts.tb;
  prof: profunditat renames ts.prof;
  ie, iep: index_expansio;
begin
  if td(idp).d.td /= dproc then raise no_es_proc; end if;
  iep:= 0; ie:= td(idp).next;
  while ie /= 0 and then te(ie).id /= ida loop
    iep:= ie; ie:= te(ie).next;
  end loop;
  error:=false;
  if ie /= 0 then error:= true; end if;
  if not error then
    ie:= tb(prof); ie:= ie+1; tb(prof):= ie;
    te(ie).id:= ida; te(ie).d:= da; te(ie).prof:= -1;
    if iep = 0 then td(idp).next:= ie;
    else te(iep).next:= ie;
    end if;
    te(ie).next:= 0;
  end if;
end put_arg;

procedure first(ts: in tsimbols; idp: in id_nom; it: out iterador_arg) is
  td: tdescripcio renames ts.td;
begin
  if td(idp).d.td /= dproc then raise no_es_proc; end if;
  it.ie:= td(idp).next;
end first;

procedure next(ts: in tsimbols; it: in out iterador_arg) is
  te: texpansio renames ts.te;
begin
  if it.ie=0 then raise mal_us; end if;
  it.ie:= te(it.ie).next;
end next;

procedure get(ts: in tsimbols; it: in iterador_arg; ida: out id_nom; da: out descripcio) is
  te: texpansio renames ts.te;
begin
  if it.ie=0 then raise mal_us; end if;
  ida:= te(it.ie).id; da:= te(it.ie).d;
end get;

function is_valid(it: in iterador_arg) return boolean is
begin
  return it.ie /= 0;
end is_valid;

procedure enter_block(ts: in out tsimbols) is
  tb: tblocks renames ts.tb;
  prof: profunditat renames ts.prof;
  ie: index_expansio;
begin
  ie:= tb(prof);
  prof:= prof+1;
  tb(prof):= ie;
end enter_block;

procedure exit_block(ts: in out tsimbols) is
  td: tdescripcio renames ts.td;
  tb: tblocks renames ts.tb;
  te: texpansio renames ts.te;
  prof: profunditat renames ts.prof;
  ie, il: index_expansio;
  id: id_nom;
begin
  ie:= tb(prof); prof:= prof-1; il:= tb(prof);
  while ie > il loop
    if te(ie).prof /= -1 then
      id:= te(ie).id;
      td(id).prof:= te(ie).prof;
      td(id).d:= te(ie).d;
      td(id).next:= te(ie).next;
    end if;
    ie:= ie-1;
  end loop;
end exit_block;
```

```
  function get_prof(ts: in tsimbols) return profunditat is
  begin
    return ts.prof;
  end get_prof;

  function "<"(prof1, prof2: in profunditat) return boolean is
  begin
    return Integer(prof1) < Integer(prof2);
  end "<";

  function value(prof: in profunditat) return integer is
  begin
    return Integer(prof);
  end value;

end decls.d_tsimbols;
```