```
------------------------------------------------------------------------------
--alphabetical character
ALPH_C [a-zA-Z]

--graphic char including " and Space
GC [\040-\176]

--idem as before excluding "
GC_NON_Q [\040-\041]|[\043-\176]

--non graphic char and Space
NON_GC [\001-\040]

--digit
D [0-9]

--non zero digit
NZD [1-9]

--numbers like x_yyy, xy_yyy, xyy_yyy, xyy(_yyy)+ where x != 0
NUM_WITH_LOWERBAR (({D}|{NZD}{D}|{NZD}{D}{D})(_({D}{D}{D}))+)

--numbers not starting with 0
NUM_WITHOUT_LOWERBAR ({NZD}{D}*)

--inner element of a string, it may be a graphic character (without the double quote) or a double
double quote.
STRING_BODY (\"\"|{GC_NON_Q}+)*

NUMBER {NUM_WITH_LOWERBAR}|{NUM_WITHOUT_LOWERBAR}

STRING \"{STRING_BODY}\"

IDENTIFIER {ALPH_C}(_?({ALPH_C}|{D}))*

CHARACTER \'{GC}\'

COMMENT \-\-{GC}*\n

OTHERS {NON_GC}+

%%

"procedure"                                     {ECHO; RETURN Pc_procedure;}
"is"                                            {ECHO; RETURN Pc_is;}
"begin"                                         {ECHO; RETURN Pc_begin;}
"end"                                           {ECHO; RETURN Pc_end;}
"const"                                         {ECHO; RETURN Pc_const;}
"new"                                           {ECHO; RETURN Pc_new;}
"type"                                          {ECHO; RETURN Pc_type;}
"record"                                        {ECHO; RETURN Pc_record;}
"array"                                         {ECHO; RETURN Pc_array;}
"of"                                            {ECHO; RETURN Pc_of;}
"while"                                         {ECHO; RETURN Pc_while;}
"loop"                                          {ECHO; RETURN Pc_loop;}
"if"                                            {ECHO; RETURN Pc_if;}
"else"                                          {ECHO; RETURN Pc_else;}
"then"                                          {ECHO; RETURN Pc_then;}
"not"                                           {ECHO; RETURN Pc_not;}
"in"                                             {ECHO; RETURN Pc_in;}
"in out"                                         {ECHO; RETURN Pc_in_out;}
"null"                                           {ECHO; RETURN Pc_null;}
"range"                                          {ECHO; RETURN Pc_range;}
"and"                                            {ECHO; RETURN Pc_and;}
"or"                                             {ECHO; RETURN Pc_or;}
"("                                              {ECHO; RETURN Parentesi_o;}
```

```
")"                                                 {ECHO; RETURN Parentesi_t;}
"."                                                 {ECHO; RETURN Punt;}
","                                                 {ECHO; RETURN Coma;}
":"                                                 {ECHO; RETURN Dospunts;}
";"                                                 {ECHO; RETURN Punticoma;}
":="                                                {ECHO; RETURN Dospuntsigual;}
"<"                                                 {ECHO;
rl_op_rel(YYLVal, tok_begin_col, tok_begin_line, d_atribut.menor);
                                                                    RETURN Op_rel;}
">"                                                 {ECHO;
rl_op_rel(YYLVal, tok_begin_col, tok_begin_line, d_atribut.major);
                                                                    RETURN Op_rel;}
"<="                                                {ECHO;
rl_op_rel(YYLVal, tok_begin_col, tok_begin_line, d_atribut.menorigual);
                                                                    RETURN Op_rel;}
">="                                                {ECHO;
rl_op_rel(YYLVal, tok_begin_col, tok_begin_line, d_atribut.majorigual);
                                                                    RETURN Op_rel;}
"="                                                 {ECHO;
rl_op_rel(YYLVal, tok_begin_col, tok_begin_line, d_atribut.igual);
                                                                    RETURN Op_rel;}
"/="                                                {ECHO;
rl_op_rel(YYLVal, tok_begin_col, tok_begin_line, d_atribut.diferent);
                                                                    RETURN Op_rel;}
"+"                                                 {ECHO; RETURN S_mes;}
"-"                                                 {ECHO; RETURN S_menys;}
"*"                                                 {ECHO; RETURN S_prod;}
"/"                                                 {ECHO; RETURN S_quoci;}
"mod"                                               {ECHO; RETURN Pc_mod;}
{IDENTIFIER}                                        {ECHO;
rl_identifier(YYLVal, tok_begin_col, tok_begin_line, yytext);
                                                                    RETURN Identif;}
{NUMBER}                                            {ECHO;
rl_literal(YYLVal, tok_begin_col, tok_begin_line, yytext);
                                                                    RETURN Lit;}
{STRING}                                            {ECHO;
rl_literal(YYLVal, tok_begin_col, tok_begin_line, yytext);
                                                                    RETURN Lit;}
{CHARACTER}                                         {ECHO;
rl_literal(YYLVal, tok_begin_col, tok_begin_line, yytext);
                                                                    RETURN Lit;}

-- els comentaris no ens importen per re?
{COMMENT}                                           {ECHO;}
{OTHERS}                                            {null;}
.                                                   {ECHO; RETURN Error;}

%%

    with Ada.Text_IO, Ada.Integer_Text_IO; use Ada.Text_IO, Ada.Integer_Text_IO;
    with pershe_tokens; use pershe_tokens;
    with pershe_io, pershe_dfa;use pershe_io, pershe_dfa;
    package a_lexic is
        procedure open(name: in String);
        procedure close;
        function YYLex return Token;    -- YYText?

        --Auxiliar functions to allow external packages use these *_dfa functions
        function YYText return String;
        function YYLength return Integer;
    end a_lexic;

    with d_atribut;
    with rutines_lexiques; use rutines_lexiques;
    package body a_lexic is
```

```
        procedure open(name: in String) is
        begin
                Open_Input(name);
        end open;

        procedure close is
        begin
                Close_Input;
        end close;
##
        --Auxiliar functions to allow external packages use these *_dfa functions

        function YYText return String is
        begin
            return pershe_dfa.YYText;
        end YYText;

        function YYLength return Integer is
        begin
            return pershe_dfa.YYLength;
        end YYLength;

    end a_lexic;

    -- DEPRECATED
    with Ada.Text_IO;  use Ada.Text_IO;
    with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
    with Ada.Command_Line; use Ada.Command_Line;
    with pershe_tokens, a_lexic; use pershe_tokens, a_lexic;
    with d_tnoms, general_defs; use d_tnoms, general_defs;
    procedure C_Tnoms is
        tk      : Token;
        tn      : tnoms;
        i,i2,j      : Integer;
        id      : general_defs.id_nom;
        ids     : array(1..50) of general_defs.id_nom; --50 son prous elements per un joc de proves
        petit
        id_st   : general_defs.id_str;
        ids_str : array(1..50) of general_defs.id_str;
    begin
        if Argument_Count=1 then
            open(Argument(1)); --per fer proves
            i:=1; i2:=1;
        --  empty(tn);
            Read_Input:
            loop
                tk:=YYLex;

                case tk is
                    when Identif =>     d_tnoms.put(tn,YYText,id); New_Line(1);ids(i):=id; i:=i+1;
                    Put_Line("ID:"&YYText);
                    when End_Of_Input => Put_Line("EOF");
                    when others => New_Line(1);
                end case;

            exit Read_Input
                when tk= End_of_Input;
            end loop Read_Input;

            Put_Line("Tnoms content:");
            j:=1;
            while j<i loop
                Put_Line("Id("&ids(j)'Img&"):"&get(tn,ids(j)));
                j:=j+1;
            end loop;
            j:=1;
```

```
            while j<i2 loop
                Put_Line("Str("&ids_str(j)'Img&"):"&get(tn,ids_str(j)));
                j:=j+1;
            end loop;
            close;
        else
            Put("Usage: ./c_tnoms <file name>.p");
        end if;
    end C_Tnoms;
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
%token Pc_procedure Pc_is Pc_begin Pc_end Pc_const Pc_new Pc_type Pc_record Pc_array Pc_of Pc_in
Pc_in_out Pc_null Pc_range Pc_while Pc_loop Pc_if Pc_then Pc_else
%right Dospuntsigual
%right Dospunts
%right Coma
%left Punt Punticoma
%right Pc_and Pc_or
%nonassoc Op_rel
%left S_mes S_menys
%left S_prod S_quoci
%left Pc_mod
%right Pc_not
%right Lit Identif
%token Parentesi_t
%right Parentesi_o -- aquest token tambe s'empra com a dummy per otorgar precedencia i
associativitat a M1

%with d_atribut;
{
subtype YYSType is d_atribut.atribut;
}

%%


PROC:
     Pc_procedure C_PROC Pc_is
        DECLS
     Pc_begin
        SENTS
     Pc_end Pc_procedure Punticoma              {rs_Proc($$,$2,$4,$6);}
  ;

DECLS:
     DECLS DECL                                 {rs_Decls($$,$1,$2);}
  |--lambda                                     {null;}
  ;

DECL:
     PROC                                       {rs_Decl($$,$1);}
  |  DECL_VAR                                   {rs_Decl($$,$1);}
  |  DECL_CONST                                 {rs_Decl($$,$1);}
  |  DECL_T                                     {rs_Decl($$,$1);}
  ;

DECL_VAR:
     LID Dospunts Identif Punticoma             {rs_Decl_Var($$,$1,$3);}
  ;

DECL_CONST:
     Identif Dospunts Pc_const Identif Dospunts IDX Punticoma       {rs_Decl_Const($$,$1,$4,$6);}
  ;
```

```
DECL_T:
     Pc_type Identif Pc_is DECL_T_CONT          {rs_Decl_T($$,$2,$4);}
  ;

DECL_T_CONT:
     Pc_new RANG Punticoma                      {rs_Decl_T_Cont($$,$2);}
  |  Pc_record
        DCAMPS
     Pc_end Pc_record Punticoma                 {rs_Decl_T_Cont($$,$2);}
  |  Pc_array Parentesi_o LID Parentesi_t Pc_of Identif Punticoma   {rs_Decl_T_Cont($$,$3,$6);}
  ;

DCAMPS:
     DCAMPS DCAMP                               {rs_DCamps($$,$1,$2);}
  |  DCAMP                                      {rs_DCamps($$,$1);}
  ;

DCAMP:
     DECL_VAR                                   {rs_DCamp($$,$1);}
  ;

C_PROC:
     Identif Parentesi_o ARGS Parentesi_t       {rs_C_Proc($$,$1,$3);}
  |  Identif                                    {rs_C_Proc($$,$1);}
  ;

ARGS:
     ARGS Punticoma ARG                         {rs_Args($$,$1,$3);}
  |  ARG                                        {rs_Args($$,$1);}
  ;

ARG:
     LID Dospunts MODE Identif                  {rs_Arg($$,$1,$3,$4);}
  ;

MODE:
     Pc_in                                      {rs_Mode($$,d_atribut.md_in);}
  |  Pc_in_out                                  {rs_Mode($$,d_atribut.md_in_out);}
  ;

LID:
     LID Coma Identif                           {rs_Lid($$,$1,$3);}
  |  Identif                                    {rs_Lid($$,$1);}
  ;

RANG:
     Identif Pc_range IDX Punt Punt IDX         {rs_Rang($$,$1,$3,$6);}
  ;

IDX:
     S_menys IDX_CONT                          {rs_Idx($$,$2,general_defs.negatiu);}
  |  IDX_CONT                                  {rs_Idx($$,$1,general_defs.positiu);}
  ;

IDX_CONT:
     Lit                                        {rs_Idx_Cont($$,$1);}
  |  Identif                                    {rs_Idx_Cont($$,$1);}
  ;

SENTS:
     SENTS_NOB                                  {rs_Sents($$,$1);}
  |  Pc_null Punticoma                          {null;}
  ;

SENTS_NOB:
     SENTS_NOB SENT                             {rs_Sents_Nob($$,$1,$2);}
```

```
  |  SENT                                    {rs_Sents_Nob($$,$1);}
  ;

SENT:
     S_ITER                                  {rs_Sent($$,$1);}
  |  S_COND                                  {rs_Sent($$,$1);}
  |  S_CRIDA                                 {rs_Sent($$,$1);}
  |  S_ASSIGN                                {rs_Sent($$,$1);}
  ;

S_ITER:
     Pc_while EXPR Pc_loop
        SENTS
     Pc_end Pc_loop Punticoma                {rs_SIter($$,$2,$4);}
  ;

S_COND:
     Pc_if EXPR Pc_then
        SENTS
     Pc_end Pc_if Punticoma                  {rs_SCond($$,$2,$4);}
  |  Pc_if EXPR Pc_then
        SENTS
     Pc_else
        SENTS
     Pc_end Pc_if Punticoma                  {rs_SCond($$,$2,$4,$6);}
  ;

S_CRIDA:
     REF Punticoma                           {rs_SCrida($$,$1);}
  ;

S_ASSIGN:
     REF Dospuntsigual EXPR Punticoma        {rs_SAssign($$,$1,$3);}
  ;

REF:
     Identif QS                              {rs_Ref($$,$1,$2);}
  ;

QS:
     QS Q                                    {rs_Qs($$,$1,$2);}
  |--lambda                                  {null;}
  ;

Q:
     Punt Identif                            {rs_Q($$,$2);}
  |  Parentesi_o LEXPR Parentesi_t           {rs_Q($$,$2);}
  ;

EXPR:
     E0                                      {rs_Expr($$,$1);}
  |  E1                                      {rs_Expr($$,$1);}
  |  E2                                      {rs_Expr($$,$1);}
  ;

E0:
     E0 Pc_and E2                            {rs_E0($$,$1,$3);}
  |  E2 Pc_and E2                            {rs_E0($$,$1,$3);}
  ;

E1:
     E1 Pc_or E2                             {rs_E1($$,$1,$3);}
  |  E2 Pc_or E2                             {rs_E1($$,$1,$3);}
  ;

E2:
```

```
     E2 Op_rel E3                                {rs_E2($$,$1,d_atribut.o_rel,$3);}
   |  E2 S_mes E3                                {rs_E2($$,$1,d_atribut.sum,$3);}
   |  E2 S_menys E3                              {rs_E2($$,$1,d_atribut.res,$3);}
   |  E2 S_prod E3                               {rs_E2($$,$1,d_atribut.prod,$3);}
   |  E2 S_quoci E3                              {rs_E2($$,$1,d_atribut.quoci,$3);}
   |  E2 Pc_mod E3                               {rs_E2($$,$1,d_atribut.modul,$3);}
   |  M1 E3 S_prod S_prod E3                     {rs_E2($$,$2,d_atribut.pot,$5);}
   |  Pc_not E3                                  {rs_E2($$,d_atribut.neg_log,$1);}
   |  S_menys E3                                 {rs_E2($$,d_atribut.neg_alg,$1);}
   |  E3                                         {rs_E2($$,$1);}
   ;

E3:
     REF                                         {rs_E3($$,$1);}
   |  Parentesi_o EXPR Parentesi_t               {rs_E3($$,$2);}
   |  Lit                                        {rs_E3($$,$1);}
   ;

M1:
     %prec Parentesi_o                           {null;}
   ;

LEXPR:
     LEXPR Coma EXPR                             {rs_LExpr($$,$1,$3);}
   |  EXPR                                       {rs_LExpr($$,$1);}
   ;

%%

    package a_sintactic is
        procedure YYParse;
    end a_sintactic;

    with d_atribut;
    with rutines_sintactiques; use rutines_sintactiques;
    package body a_sintactic is
##
    end a_sintactic;
--------------------------------------------------------------------------------



--------------------------------------------------------------------------------
with d_tnoms; use d_tnoms;
with general_defs; use general_defs;
package d_atribut is
    --pragma pure;

    type node;
        type atribut is access node;
    type pnode is access node;
        type tnode is (nd_null,
                                    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_qs,
                                nd_q,
                                nd_expr,
                                nd_e0,
                                nd_e1,
                                nd_e2,
                                nd_e3,
                                nd_lexpr,
                                nd_id,
                                nd_lit,
                                nd_op_rel);

 type trelacio is (menor,major,menorigual,majorigual,igual,diferent);
 type tmode is (md_in,md_in_out);
 type posicio is
     record
         fila: natural;
         columna: natural;
     end record;
type rang is
             record
                     id: id_nom;
                     linf: pnode;
                     lsup: pnode;
             end record;
     type operand is (nul, o_rel, 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_id              =>
                             id_id: id_nom;
                             id_pos: posicio;

                     when nd_lit             =>
                             lit_ids: id_str;
                             lit_pos: posicio;

                     when nd_op_rel  =>
                             orel_tipus: trelacio;
                             orel_ope:pnode;
                             orel_opd: pnode;

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

               when nd_mode    =>
                     mode_tipus: tmode;
```

```
when nd_c_proc  =>
      cproc_id: pnode;
      cproc_args: pnode;

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

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_id: 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_rang: rang;

  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: id_nom;
                        ref_qs: pnode;

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

                    when nd_q =>
                        q_contingut: pnode;

                    when nd_expr =>
                        expr_e: pnode;

                    when nd_e0 | nd_e1 =>
                        e_ope: pnode;
                        e_opd: pnode;

                    when nd_e2 =>
                        e2_ope: pnode;
                        e2_opd: pnode;
                        e2_operand: operand;

                    when nd_e3 =>
                        e3_cont: pnode;

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

            end case;
        end record;

end d_atribut;
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
package general_defs is
    pragma pure;
```

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

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

    null_id: constant id_nom:=0;


-- T_Simbols stuff
        type tidx is (positiu, negatiu);
        type valor is
                record
                        signe: tidx;
                        v: natural; --temp
                end record;
        type num_var is new natural; -- sols per compilar
        type num_proc is new natural; -- sols per compilar


    type despl is new natural;

    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; -- tipus dels components de l'arry
                                        -- other gc stuff
                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; -- gc
                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; -- gc
                when dproc  =>
                                        np: num_proc; -- gc
                when dargc  =>
                                        ta: id_nom; -- tipus de l'argument
                                        na: num_var; -- gc
            end case;
        end record;

end general_defs;
--------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------
with d_atribut; use d_atribut;
package rutines_lexiques is
    -- pragma pure;

    procedure rl_identifier(a: out atribut; col: in natural; fil: in natural; text: in String );
    procedure rl_literal(a: out atribut; col: in natural; fil: in natural; text: in String);
    procedure rl_op_rel(a: out atribut; col: in natural; fil: in natural; tr: in trelacio);

end rutines_lexiques;
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
with d_tnoms; use d_tnoms;
with general_defs; use general_defs;
with d_atribut; use d_atribut;
package body rutines_lexiques is
    --pragma pure;
    --Definicions
    tn: tnoms; -- Creo que esto deberia ir en un paquete aparte, donde?!


    --Procedimientos
    procedure rl_identifier(a: out atribut; col: in natural; fil: in natural; text: in String) is
        pos: posicio;
        id: id_nom;
    begin
        a:= new node(nd_id);

        pos:= (fil,col);
        put(tn,text,id);

        a.id_pos:= pos; a.id_id:= id;
    end rl_identifier;


    procedure rl_literal(a: out atribut; col: in natural; fil: in natural; text: in String) is
        pos: posicio;
        ids: id_str;
    begin
        a:= new node(nd_lit);

        pos:= (fil,col);
        put(tn,text,ids);

        a.lit_pos:= pos; a.lit_ids:= ids;
    end rl_literal;


    procedure rl_op_rel(a: out atribut; col: in natural; fil: in natural; tr: in trelacio) is
    begin
        a:= new node(nd_op_rel);

        a.orel_tipus:= tr; a.orel_ope:= null; a.orel_opd:= null;
    end rl_op_rel;

end rutines_lexiques;
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
package rutines_sintactiques is
```

```
    -- Procediment
    procedure rs_Proc(proc: out YYSType; cproc: in YYSType; decls: in YYSType; sents: in YYSType);
    procedure rs_C_Proc(cproc: out YYSType; proc_id: in YYSType; args: in YYSType);
    procedure rs_C_Proc(cproc: out YYSType; proc_id: in YYSType);
    procedure rs_Args(args: out YYSType; args_seg: in YYSType; arg: in YYSType);
    procedure rs_Args(args: out YYSType; arg: in YYSType);
    procedure rs_Arg(arg: out YYSType; lid: in YYSType; mode: in YYSType; tipus: in YYSType);
    procedure rs_Mode(mode: out YYStype; tipus: in tmode);


    -- Declaracio
    procedure rs_Decls(decls: out YYSType; decls_seg: in YYSType; decl: YYSType);
    procedure rs_Decl(decl: out YYSType; decl_real: in YYSType);
    procedure rs_Decl_Var(decl: out YYSType; lista_id: in YYSType; tipus: in YYSType);
    procedure rs_Decl_Const(decl: out YYSType; id_const: in YYSType; tipus: in YYSType; valor: in
    YYSType);
    procedure rs_Idx(idx: out YYSType; idx_cont: in YYSType; signe: in tidx);
    procedure rs_Idx_Cont(idx_cont: out YYSType; valor: in YYSType);
    procedure rs_Decl_T(decl: out YYSType; id_type: in YYSType; decl_cont: in YYSType);
    procedure rs_Decl_T_Cont(decl: out YYSType; info: in YYSType);
    procedure rs_Decl_T_Cont(decl: out YYSType; rang_array: in YYSType; tipus_array: in YYSType);
    procedure rs_DCamps(camps: out YYSType; camp_seg: in YYStype; camp: in YYStype);
    procedure rs_DCamps(camps: out YYSType; camp: in YYStype);
    procedure rs_DCamp(camp: out YYStype; var: in YYStype);


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


    -- Expressio
    procedure rs_LExpr(lexpr: out YYStype; cont: in YYStype; expr: in YYStype);
    procedure rs_LExpr(lexpr: out YYStype; expr: in YYStype);
    procedure rs_Expr(expr: out YYStype; cont: in YYStype);
    procedure rs_E0(expr: out YYStype; ee: in YYStype; ed: in YYStype);
    procedure rs_E1(expr: out YYStype; ee: in YYStype; ed: in YYStype);
    procedure rs_E2(expr: out YYStype; ee: in YYStype; op: in operand; ed: in YYStype);
    procedure rs_E2(expr: out YYStype; op:in operand; ed: in YYStype);
    procedure rs_E2(expr: out YYStype; ed: in YYStype);
    procedure rs_E3(expr: out YYStype; e: in YYStype);


    -- Altres
    procedure rs_Lid(lid: out YYStype; id_seg: in YYStype; id: in YYStype);
    procedure rs_Ref(ref: out YYSType; ref_id: in YYSType; qs: in YYSType);
    procedure rs_Qs(qs: out YYSType; qs_in: in YYSType; q: in YYSType);
    procedure rs_Q(q: out YYSType; contingut: in YYSType);


    error,type_error,proc_error,arg_error,record_error,camp_error,array_error,var_error,const_error
    : exception;
end rutines_sintactiques;
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
```

```
use d_atribut; with d_atribut;
use general_defs; with general_defs;
use d_tsimbols; with d_tsimbols;
package body rutines_sintactiques is
    ts: tsimbols; -- Lo mismo que la de nombres!!


    -- Procediment

    procedure rs_Proc(proc: out YYSType; cproc: in YYSType; decls: in YYSType; sents: in YYSType)
    is
        desc: descripcio;
        id: id_nom;
        error: boolean;
        p: pnode;
    begin
        proc:= new node(nd_proc);
        proc.proc_cproc:= cproc;
        proc.proc_decl:= decls;
        proc.proc_sents:= sents;

        id:= cproc.cproc_id.id_id;
        desc:= new descripcio(dproc);
        desc.np:=nou_proc;

        put(ts,id,desc,error);
        if error then raise proc_error; end if;

        putargs(args,id);
    end rs_Proc;


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


    procedure rs_C_Proc(cproc: out YYSType; proc_id: in YYSType) is
    begin
        cproc:= new node(nd_c_proc);
        cproc.cproc_id:= proc_id;
        cproc.cproc_args:= null;
    end rs_C_Proc;


    procedure rs_Args(args: out YYSType; args_seg: in YYSType; arg: in YYSType) is
    begin
        args:= new node(nd_args);
        args.args:= args_seg;
        args.arg:= arg;
    end rs_Args;


    procedure rs_Args(args: out YYSType; arg: in YYSType) is
    begin
        args:= new node(nd_args);
        args.args:= null;
        args.arg:= arg;
    end rs_Args;


    procedure rs_Arg(arg: out YYSType;lid: in YYSType; mode: in YYSType; tipus: in YYSType) is
        desc: descripcio;
```

```
begin
    arg:= new node(nd_arg);
    arg.lid:= lid;
    arg.arg_mode:= mode.tmode;
    arg.arg_tipus:= tipus;
end rs_Arg;


procedure rs_Mode(mode: out YYStype; tipus: in tmode) is
begin
    mode:= new node(nd_mode);

    mode.mode_tipus:= tipus;
end rs_Mode;


-- Declaracio

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


procedure rs_Decl(decl: out YYSType; decl_real: in YYSType) is
begin
    decl.decl_real:= decl_real;
end rs_Decl;


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


procedure rs_Decl_Const(decl: out YYSType; id_const: in YYSType; tipus: in YYSType; valor: in
YYSType) is
    desc: descripcio;
    error: boolean;
begin
    decl:= new node(nd_decl_const);
    decl.dconst_id:= id_const;
    decl.dconst_tipus:= tipus;
    decl.dconst_valor:= valor;

    desc:= new descripcio(dconst);
    desc.tc:= tipus.id_id;
    --desc.valor:=??

    put(ts,id_const.id,desc,error);
    if error then raise const_error; end if;
end rs_Decl_Const;


procedure rs_Idx(idx: out YYSType; idx_cont: in YYSType; signe: in idx) is
begin
    idx:= new node(nd_idx);
    idx.idx_cont:= idx_cont;
    idx.idx_tipus:= signe;
end rs_Idx;
```

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


procedure rs_Decl_T(decl: out YYSType; id_type: in YYSType; decl_cont: in YYSType) is
    desc: descripcio;
    id: id_nom;
    error: boolean;
begin
    decl:= new node(nd_decl_t);
    decl.dt_id:= id_type;
    decl_dt_cont:= decl_cont;

    id:= id_type.id_id;
    desc:= new descripcio(dtipus);

    case decl_cont.tn is
        when nd_decl_t_cont_type    =>
            desc.dt:= new descr_tipus(tsb_null); --Deberia ser ent/car/bool
            --desc.dt.ocup:=??;
            put(ts,id,desc,error);
            if error then raise type_error; end if;

        when nd_decl_t_cont_record  =>
            desc.dt:= new descr_tipus(tsb_rec);
            --desc.dt.ocup:=??

            put(ts,id,desc,error);
            if error then raise record_error; end if;
            putcamps(decl_cont.dtcont_camps,id);

        when nd_decl_t_cont_arry    =>
            desc.dt:= new descr_tipus(tsb_arr);
            desc.tcomp:= decl_cont.dtcont_tipus.id_id;
            --desc.dt.ocup:= ??;

            put(ts,id,desc,error);
            if error then raise array_error; end if;
            putindxs(decl_cont.dtcont_idx,id);

    end case;

end rs_Decl_T;


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

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

    end case;
end rs_Decl_T_Cont;


procedure rs_Decl_T_Cont(decl: out YYSType; rang_array: in YYSType; tipus_array: in YYSType) 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 YYSType; camp_seg: in YYStype; camp: in YYStype) is
begin
    camps:= new node(nd_dcamps);

    camps.dcamps_dcamps:= camp_seg;
    camps.dcamps_dcamp:= camp;
end rs_DCamps;


procedure rs_DCamps(camps: out YYSType; camp: in YYStype) is
begin
    camps:= new node(nd_dcamps);

    camps.dcamps_dcamps:= null;
    camps.dcamps_dcamp:= camp;
end rs_DCamps;



procedure rs_DCamp(camp: out YYStype; var: in YYStype) is
begin
    camp:= new node(nd_dcamp);

    camp.dcamp_decl:= var;
end rsDCamp;


-- Sentencia


procedure rs_Sents(sents: out YYStype; sent: in YYStype) is
begin
    sents:= new node(nd_sents);

    sents.sents_cont:= sent;
end rs_Sents;


procedure rs_Sent_Nob(sents: out YYStype; sent_cont: in YYSType; sent: in YYStype) 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 YYStype; sent: in YYStype) is
begin
    sents:= new node(nd_sents_nob);

    sents.snb_snb:= null;
    sents.snb_sent:= sent;
end rs_Sent_Nob;


procedure rs_Sent(sent: out YYStype; stipus: in YYStype) is
begin
    sent:= new node(nd_sent);
```

```
    sent.sent_sent:= stipus;
end rs_Sent;


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


procedure rs_SCond(sent: out YYStype; expr: in YYStype; sents: in YYStype) is
begin
    sent:= new node(nd_scond);

    sent.scond_expr:= expr;
    sent.scond_sents:= sents;
    sent.scond_esents:= null;
end rs_SCond;


procedure rs_SCond(sent: out YYStype; expr: in YYStype; sents_if: in YYStype; sents_else: in
YYStype) 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 YYStype; ref: in YYStype) is
begin
    sent:= new node(nd_scrida);

    sent.scrida_ref:= ref;
end rs_SCrida;


procedure rs_SAssign(sent: out YYStype; ref: in YYStype; expr: in YYStype) is
begin
    sent:= new node(nd_sassign);

    sent.sassign_ref:= ref;
    sent.sassign_expr:= expr;
end rs_SAssign;


-- Expressio

procedure rs_LExpr(lexpr: out YYStype; cont: in YYStype; expr: in YYStype) is
begin
    lexpr:= new node(nd_lexpr);

    lexpr.lexpr_cont:= cont;
    lexpr.lexpr_expr:= expr;
end rs_LExpr;


procedure rs_LExpr(lexpr: out YYStype; expr: in YYStype) is
begin
    lexpr:= new node(nd_lexpr);
```

```
    lexpr.lexpr_cont:= null;
    lexpr.lexpr_expr:= expr;
end rs_LExpr;


procedure rs_Expr(expr: out YYStype; cont: in YYStype) is
begin
    expr:= new node(nd_expr);

    expr.expr_e:= cont;
end rs_Expr;


procedure rs_E0(expr: out YYStype; ee: in YYStype; ed: in YYStype) is
begin
    expr:= new node(nd_e0);

    expr.e_ope:= ee;
    expr.e_opd:= ed;
end rs_E0;


procedure rs_E1(expr: out YYStype; ee: in YYStype; ed: in YYStype) is
begin
    expr:= new node(nd_e1);

    expr.e_ope:= ee;
    expr.e_opd:= ed;
end rs_E1;


procedure rs_E2(expr: out YYStype; ee: in YYStype; op: in operand; ed: in YYStype) is
begin
    expr:= new node(nd_e2);

    expr.e2_ope:= ee;
    expr.e2_opd:= ed;
    expr.e2_operand:= op;
end rs_E2;


procedure rs_E2(expr: out YYStype; op:in operand; ed: in YYStype) is
begin
    expr:= new node(nd_e2);

    expr.e2_ope:= null;
    expr.e2_opd:= ed;
    expr.e2_operand:= op;
end rs_E2;


procedure rs_E2(expr: out YYStype; ed: in YYStype) is
begin
    expr:= new node(nd_e2);

    expr.e2_ope:= null;
    expr.e2_opd:= null;
    expr.e2_operand:= op;
end rs_E2;


procedure rs_E3(expr: out YYStype; e: in YYStype) is
begin
    expr:= new node(nd_e3);

    expr.e3_cont:= e;
```

```
end rs_E3;


-- Altres

procedure rs_Lid(lid: out YYStype; id_seg: in YYStype; id: in YYStype) is
begin
    lid:= new node(nd_lid);

    lid.lid_seg:= id_seg;
    lid.lid_id:= id;
end rs_Lid;


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

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


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


--rutines auxiliars

procedure putvar(list_id: in YYSType; id_type: in id_nom) is
    desc: descripcio;
    p,ps: pnode;
    error: boolean;
begin
    p:= list_id.lid_id;
    ps:= list_id.lid_next
    desc:= new descripcio(dvar);
    desc.tv:= id_type;
    desc.nv:= nova_var;
    put(ts,p.id_id,desc,error);
    if error then raise var_error; end if;

    if ps/=null then
        putvar(ps,id_type);
    end if;
end putvar;


procedure putcamps(camps: in YYSType; idr: in id_nom) is
    desc: descripcio;
    p, ps: pnode;
    error: boolean;
begin
   desc:= new descripcio(dcamp);
   desc.tcmp:= p.dcamp_decl.dvar_tipus.id_id;
   --desc.dcmp:= ??
```

```
   --Posar tots el camps que estan en la mateixa
   --llista de variables (LID)
   p:=camps.dcamps_dcamp.dcamp_decl.dvar_lid;
   while p/=null loop
       put_camp(ts,idr,p.lid_id.id_id,desc,error);
       if error then raise camp_error; end if;
       p:= p.lid_seg;
   end loop;

   --Seguir amb la llista de camps
   ps:= camps.dcamps_dcamps;
   if ps/=null then
       putcamps(ps,idr);
   end if;
end putcamps;


procedure putargs(args: in YYSType; idp: id_nom) is
    p,ps: pnode;
    error: boolean;
    desc: descripcio;
begin
    p:= args.arg;

    case p.arg_mode is
        when md_in  =>
            desc:= new descripcio(dargc);
            desc.ta:= p.arg_tipus.id_id;
            desc.na:= nova_var;

        when md_in_out  =>
            desc:= new descripcio(dvar);
            desc.tv:= p.arg_tipus.id_id;
            desc.nv:= nova_var;
    end case;

    p:= p.arg_lid;
    while p/=null loop
       put_arg(ts,idp,p.lid_id.id_id,desc,error);
       if error then raise arg_error; end if;
       p:= p.lid_seg;
    end loop;

    ps:= args.args;
    if ps/=null then
        putargs(ps,idp);
    end if;
end putargs;


procedure putindxs(indxs: in YYSType; ida: in id_nom) is
    desc: descripcio;
begin
    p:= indxs.lid_id;
    ps:= indxs.lid_seg;

    id:= p.id_id;
    desc:= new descripcio(dindx);
    desc.tind:= id;
    put_index(ts,ida,desc);

    if ps/= null then
        putindxs(ps,ida);
    end if;
end putindxs;
```

```
    -- Temporal, a la generacio de codi canvien.
    -- Añadirlos como variables&procs a general_defs?
    nv: num_var:= 0;
    np: num_proc:= 0;

    function nova_var return Num_var is
    begin
        nv:= nv+1;
        return nv;
    end nova_var;

    function nou_proc return Num_proc is
    begin
        np:= np+1;
        return np;
    end nou_proc;

end rutines_sintactiques;
--------------------------------------------------------------------------------



--------------------------------------------------------------------------------
with Ada.Containers; use Ada.Containers;
with Ada.Strings; use Ada.Strings;
with general_defs; use general_defs;
package d_tnoms is
    pragma pure;

    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:= (others=> (null_id, 0));
            ts: str_table:= (others=> max_ch);
```

```
            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 d_tnoms;
-------------------------------------------------------------------------------------------------
-------------


-------------------------------------------------------------------------------
with Ada.Strings.Hash; use Ada.Strings;
package body 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;
    begin
        i:= hash(nom) mod b; p:= td(i);
        while p/=null_id and then not equal(nom,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, nom, 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;
    --exception
    --   when space_overflow=> put("Space overflow");
    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;--number of stored strings
        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 d_tnoms;
--------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------
with general_defs; use general_defs;
package d_tsimbols is
    pragma pure;

    type tsimbols is limited private;

    type iterador_index is private;
    type iterador_arg 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;


    -- Operacions de record
    procedure put_camp(ts: in out tsimbols; idr,idc: in id_nom; d: in descripcio; error: out
    boolean); --idr: id record , idc: id camp
    function get_camp(ts: in tsimbols; idr,idc: in id_nom) return descripcio;
    procedure update(ts: in out tsimbols; id: in id_nom; d: in 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: in 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_bock(ts: in out tsimbols);


    no_es_tipus, no_es_record, no_es_array, no_es_proc, mal_us: exception;

private

    type index_expansio is integer range 0..max_id;

    type te_item;
    type td_item is
        record
            prof: integer;
            d: descripcio;
            next: index_expansio;
        end record;

    type te_item is
        record
            id: id_nom;
            prof: integer;
            d: descripcio;
```

```
            next: index_expansio;
        end record;

    --!!!los rangos no estan bien puestos!!!
    type tdescripcio is array (id_nom) of td_item;
    type texpansio is array (index_expansio) of te_item;
    type tblocks is array (id_nom) of index_expansio;

    type tsimbols is
        record
            prof: integer;
            td: tdescripcio;
            te: texpansio;
            tb: tblocks;
        end record;


    type iterador_index is index_expansio;
    type iterador_arg is index_expansio;

end d_tsimbols;
-------------------------------------------------------------------------------



-------------------------------------------------------------------------------
with general_defs; use general_defs;
package body d_tsimbols is
    pragma pure;


    procedure empty(ts: out tsimbols) is
        td: tdescripcio renames ts.td;
        tb: tblocks renames ts.tb;
        prof: integer renames ts.prof;
    begin
        for id in id_nom loop
            td(id):= (0,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: integer 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;
            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; d: in descripcio; error: out
boolean) is
    td: tdescripcio renames ts.td;
    te: texpansio renames ts.te;
    prof: integer 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(id).next; td(id).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:= dnula;
            else d:= te(ie).d;
    end if;
    return d;


procedure update(ts: in out tsimbols; id: in id_nom; d: in descripcio) is
    td: tdescripcio renames ts.td;
    tb: tblocks renames ts.tb;
    te: texpansio renames ts.te;
    prof: integer renames ts.prof;
    ie: index_expansio;
    ocup: despl;
begin
    if td(id).d.td /= dtipus then raise no_es_tipus; end if;
    if td(id).d.dt.tsb /= tsb_rec then
        raise no_es_record;
    end if;
    ie:= td(id).next;
    ocup:= 0;
    while ie/=0 loop
        ocup:= ocup+te(ie).d.dcmp;
    end loop;
    td(id).d.dt.ocup:= ocup;
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: integer renames ts.prof;
    ie: 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: in 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:= ts(id).next;
end first;


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


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


function is_valid(it: in iterador_index) return boolean is
begin
    return it /= 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;
    prof: integer renames td.prof;
    ie, iep: index_expansio;
begin
    if td(idp).d.td /= dproc then raise no_es_proc; end if;
    iep:= 0; ie:= td(id).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(id).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:= td(idp).next;
end first;


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


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


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


procedure exit_bock(ts: in out tsimbols) is
    td: tdescripcio renames ts.td;
    tb: tblocks renames ts.tb;
    te: texpansio renames ts.te;
    prof: integer 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_bock;

end d_tsimbols;
--------------------------------------------------------------------------------
```