STRUKTUR POHON
DALAM BAHASA PROLOG

Struktur Pohon diimplementasikan sebagai struktur komposisi (tipe bentukan)

Jika struktur pohon akan diimplementasikan dalam bahasa PROLOG sebagai struktur komposisi, maka struktur pohon itu akan dibentuk sebagai tipe bernama pohon dengan bantuan functor.

Pendefinisian tipe pohon tersebut pada DOMAINS, jika menggunakan pemroses bahasa berupa compiler (contoh: Turbo Prolog). Jika menggunakan pemroses bahasa berupa interpreter, maka tipe tidak dapat didefinisikan/dideklarasikan sebelumnya melainkan langsung digunakan.

Sebagai contoh, digunakan struktur pohon berikut ini yang merupakan pohon biner, yang direalisasikan dalam bahasa pemrograman TURBO PROLOG.

Contoh Kasus :

TYPE POHON BINER - PREFIX : tree(elemen,L:tree,R:tree)

REALISASI DALAM TURBO PROLOG

```
/*-------------------------------------------------*/
/* STRUKTUR KOMPOSISI : POHON BINER                */
/*   dengan basis pohon kosong (basis-0)           */
/*-------------------------------------------------*/

DOMAINS
  /* definisi tipe pohon biner */
  dtree   =
  /* basis */
            void;
  /* rekurens */
            tree(delemen,dtree,dtree)
  /* definisi tipe dari elemen pohon biner */
  delemen = symbol
  /* definisi dari list & elemen list */
  dlist   = delemen*

PREDICATES
  concate(dlist,dlist,dlist)
  unerLeft(dtree)
  unerRight(dtree)
  isBinTree(dtree)
  isTreeMember(delemen,dtree)
  nbElmt(dtree,integer)
  nbDaun(dtree,integer)
  preorder(dtree,dlist)
  inorder(dtree,dlist)
  postorder(dtree,dlist)

CLAUSES
  /* concate(Xs,Ys,Zs) benar, jika Zs adalah hasil
     konkatenasi list Xs dan Ys */
  concate([],Ys,Ys).                                /* basis-0 */
  concate([X|Xs],Ys,[X|Zs]) :- concate(Xs,Ys,Zs).  /* rekurens */
```

```
/* unerLeft(P) benar, jika P tidak kosong adalah pohon unerleft:
   hanya mempunyai subpohon kiri */
  unerLeft(tree(_,L,void)).

/* unerRight(P) benar, jika P tidak kosong adalah pohon unerright:
   hanya mempunyai subpohon kanan */
  unerRight(tree(_,void,R)).

/* isBinTree(P) benar, jika P tidak kosong adalah pohon biner:
   mempunyai subpohon kiri dan kanan */
  isBinTree(tree(_,L,R).

/* isTreeMember(X,P) benar, jika X adalah elemen dari pohon biner P
   yang tidak kosong */
  isTreeMember(X,tree(X,_,_)).                          /* basis-1 */
  isTreeMember(X,tree(Y,L,_)) :- X <> Y, isTreeMember(X,L). /*rek*/
  isTreeMember(X,tree(Y,_,R)) :- X <> Y, isTreeMember(X,R). /*rek*/

/* nbElmt(P,N) benar, jika N adalah jumlah elemen
     (simpul) dari pohon biner P */
  nbElmt(void,0).                              /* basis-0 */
  nbElmt(tree(_,L,R),N) :- nbElmt(L,NL),    /* rekurens */
                           nbElmt(R,NR),
                           N = 1 + NL + NR.

  /* nbDaun(P,N) benar, jika N adalah jumlah daun
     dari pohon biner P */
  nbDaun(void,0).                              /* basis-0 */
  nbDaun(tree(_,void,void),1).                 /* kasus khusus */
  nbDaun(tree(_,L,R),N) :- nbDaun(L,NL),    /* rekurens */
                           nbDaun(R,NR),
                           N = NL + NR.

  /* preorder(P,Ls) benar, jika Ls adalah list of
     element (simpul) dari pohon P secara preorder */
  preorder(void,[]).                                /* basis-0 */
  preorder(tree(A,L,R),Xs) :- preorder(L,Ls),    /* rekurens */
                              preorder(R,Rs),
                              concate([A|Ls],Rs,Xs).

  /* postorder(P,Ls) benar, jika Ls adalah list of
     element (simpul) dari pohon P secara postorder */
  postorder(void,[]).                               /* basis-0 */
  postorder(tree(A,L,R),Xs) :- postorder(L,Ls),  /* rekurens */
                               postorder(R,Rs),
                               concate(Rs,[A],Rs1),
                               concate(Ls,Rs1,Xs).

  /* inorder(P,Ls) benar, jika Ls adalah list of
     element (simpul) dari pohon P secara inorder */
  inorder(void,[]).                                 /* basis-0 */
  inorder(tree(A,L,R),Xs) :- inorder(L,Ls),      /* rekurens */
                             inorder(R,Rs),
                             concate(Ls,[A|Rs],Xs).
```