Praktikum StrukturData
Modul PSDA

PSDA-13. ADT POHON BINER

1. Buatlah ADT Pohon Biner dalam Bahasa C sesuai spesifikasi di bawah ini. Untuk ADT list yang diperlukan dalam beberapa primitif, gunakan ADT List Linier pada modul P-10. Bagian 1. Jika terjadi konflik penamaan, misalnya nama type address, maka lakukan perubahan yang diperlukan.
2. Buatlah driver untuk memeriksa apakah semua primitif sudah berjalan dengan baik.

```
{ ADT Pohon Biner }

{ Modul lain yang digunakan : }
USE ADT_LIST_LINIER

{ *** Definisi Type Pohon Biner *** }
type node : < Info : infotype,
              Left : address,
              Right : address >
type BinTree : address

{ *** Definisi Type List of Node *** }
type ListOfNode : List

{ *** PROTOTYPE *** }

{ *** Selektor *** }
function GetAkar (P : BinTree) → infotype
{ Mengirimkan nilai Akar pohon biner P }
function GetLeft (P : BinTree) → BinTree
{ Mengirimkan Anak Kiri pohon biner P }
function GetRight (P : BinTree) → BinTree
{ Mengirimkan Anak Kanan pohon biner P }

{ *** Konstruktor *** }
function Tree (Akar : infotype, L : BinTree, R : BinTree) → BinTree
{ Menghasilkan sebuah pohon biner dari A, L, dan R, jika alokasi berhasil }
{ Menghasilkan pohon kosong (Nil) jika alokasi gagal }
procedure MakeTree (input Akar : infotype, input L : BinTree,
                    input R : BinTree, output P : BinTree)
{ I.S. Sembarang }
{ F.S. Menghasilkan sebuah pohon P }
{ Menghasilkan sebuah pohon biner P dari A, L, dan R, jika alokasi berhasil }
{ Menghasilkan pohon P yang kosong (Nil) jika alokasi gagal }
procedure BuildTree (output P : BinTree)
{ Membentuk sebuah pohon biner P dari pita karakter yang dibaca }
{ I.S. Pita berisi "konstanta" pohon dalam bentuk prefix.
       Memori pasti cukup, alokasi pasti berhasil. }
{ F.S. P dibentuk dari ekspresi dalam pita }

{ *** Predikat-Predikat Penting *** }
function IsTreeEmpty (P : BinTree) → boolean
{ Mengirimkan true jika P adalah pohon biner kosong }
function IsOneElmt (P : BinTree) → boolean
{ Mengirimkan true jika P adalah pohon biner tidak kosong dan hanya memiliki 1
```

```
elemen }
function IsUnerLeft (P : BinTree) → boolean
{ Mengirimkan true jika pohon biner tidak kosong P adalah pohon unerleft: hanya
mempunyai subpohon kiri }
function IsUnerRight (P : BinTree) → boolean
{ Mengirimkan true jika pohon biner tidak kosong P adalah pohon unerright: hanya
mempunyai subpohon kanan}
function IsBiner (P : BinTree) → boolean
{ Mengirimkan true jika pohon biner tidak kosong P adalah pohon biner: mempunyai
subpohon kiri dan subpohon kanan}
{ *** Traversal *** }
procedure PrintPreorder (input P : BinTree)
{ I.S. P terdefinisi }
{ F.S. Semua simpul P sudah dicetak secara preorder: akar, pohon kiri, dan pohon
kanan. Setiap pohon ditandai dengan tanda kurung buka dan kurung tutup (). }
procedure PrintInorder (input P : BinTree)
{ I.S. P terdefinisi }
{ F.S. Semua simpul P sudah dicetak secara inorder: pohon kiri, akar, dan pohon
kanan. Setiap pohon ditandai dengan tanda kurung buka dan kurung tutup ().  }
procedure PrintPostorder (input P : BinTree)
{ I.S. P terdefinisi }
{ F.S. Semua simpul P sudah dicetak secara postorder: pohon kiri, pohon kanan, dan
akar. Setiap pohon ditandai dengan tanda kurung buka dan kurung tutup ().  }
procedure PrintTree (input P : BinTree, input h : integer)
{ I.S. P terdefinisi, h adalah jarak indentasi }
{ F.S. Semua simpul P sudah ditulis dengan indentasi }

{ *** Searching *** }
function Search (P : BinTree, X : infotype) → boolean
{ Mengirimkan true jika ada node dari P yang bernilai X }

{ *** Fungsi-Fungsi Lain *** }
function NbElmt (P : BinTree) → integer
{ Mengirimkan banyaknya elemen (node) pohon biner P }
function NbDaun (P : BinTree) → integer
{ Mengirimkan banyaknya daun (node) pohon biner P }
function IsSkewLeft (P : BinTree) → boolean
{ Mengirimkan true jika P adalah pohon condong kiri }
function IsSkewRight (P : BinTree) → boolean
{ Mengirimkan true jika P adalah pohon condong kiri }
function Level (P : BinTree, X : infotype) → integer
{ Mengirimkan level dari node X yang merupakan salah satu simpul dari pohon biner
P. Akar(P) level-nya adalah 1. Pohon P tidak kosong. }

{ *** Operasi lain *** }
procedure AddDaunTerkiri (input/output P : BinTree, input X : infotype)
{ I.S. P boleh kosong }
{ F.S. P bertambah simpulnya, dengan X sebagai simpul daun terkiri }
procedure AddDaun (input/Output P : BinTree, input X, Y : infotype,
                   input Kiri : boolean)
{ I.S. P tidak kosong, X adalah salah satu daun Pohon Biner P }
{ F.S. P bertambah simpulnya, dengan Y sebagai anak kiri X (jika Kiri = true), atau
       sebagai anak Kanan X (jika Kiri = false) }
procedure DelDaunTerkiri (input/output P : BinTree, output X : infotype)
{ I.S. P tidak kosong }
{ F.S. P dihapus daun terkirinya, dan didealokasi, dengan X adalah info yang semula
```

```
        disimpan pada daun terkiri yang dihapus }
procedure DelDaun (input/output P : BinTree, input X : infotype)
{ I.S. P tidak kosong, X adalah salah satu daun }
{ F.S. Semua daun bernilai X dihapus dari P }
function MakeListDaun (P : BinTree) → ListOfNode
{ Jika P adalah pohon kosong, maka menghasilkan list kosong. }
{ Jika P bukan pohon kosong : menghasilkan list yang elemennya adalah semua daun
pohon P, jika semua alokasi list berhasil. Menghasilkan list kosong jika ada
alokasi yang gagal. }
function MakeListPreorder (P : BinTree) → ListOfNode
{ Jika P adalah pohon kosong, maka menghasilkan list kosong. }
{ Jika P bukan pohon kosong: menghasilkan list yang elemennya adalah semua elemen
pohon P dengan urutan Preorder, jika semua alokasi berhasil. Menghasilkan list
kosong jika ada alokasi yang gagal. }
function MakeListLevel (P : BinTree, N : integer) → ListOfNode
{ Jika P adalah pohon kosong, maka menghasilkan list kosong. }
{ Jika P bukan pohon kosong: menghasilkan list yang elemennya adalah semua elemen
pohon P yang levelnya=N, jika semua alokasi berhasil. Menghasilkan list kosong jika
ada alokasi yang gagal. }
{ *** Balanced tree *** }
function BuildBalanceTree (n : integer) → BinTree
{ Menghasilkan sebuah balanced tree dengan n node, nilai setiap node dibaca }

{ *** Binary Search Tree *** }
function BSearch (P : BinTree, X : infotype) → boolean
{ Mengirimkan true jika ada node dari P yang bernilai X }
function InsSearch (P : BinTree, X : infotype) → BinTree
{ Menghasilkan sebuah pohon Binary Search Tree P dengan tambahan simpul X. Belum ada
simpul P yang bernilai X. }
procedure DelBtree (input/output P : BinTree, input X : infotype)
{ I.S. Pohon P tidak kosong }
{ F.S. Nilai X yang dihapus pasti ada }
{ Sebuah node dg nilai X dihapus }
```