Pohon Biner – Bagian 2

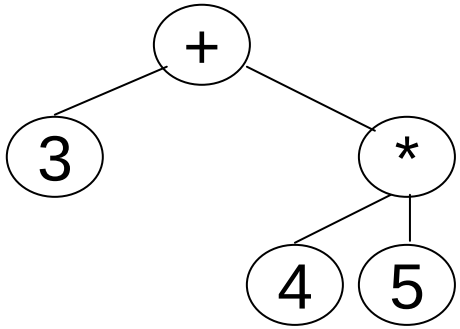
(Pohon Seimbang, Pohon Biner Terurut, Pembangunan Pohon Biner dari Pita Karakter/String)

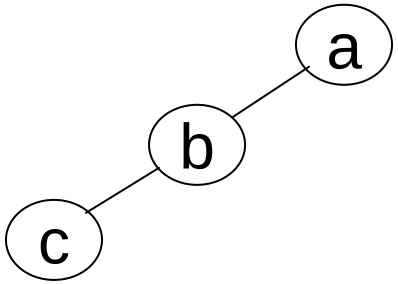
Tim Pengajar IF2030

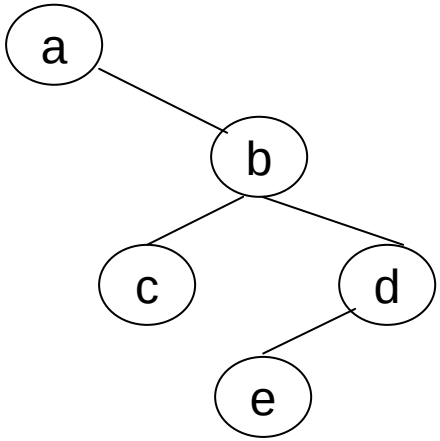
Semester I/2009-2010

Pohon Biner

- Pohon biner adalah himpunan terbatas yang
  - mungkin **kosong**, atau
  - terdiri atas sebuah simpul yang disebut **akar** dan dua buah himpunan lain yang *disjoint* yang merupakan pohon biner, yang disebut sebagai **sub pohon kiri** dan **sub pohon kanan** dari pohon biner tersebut







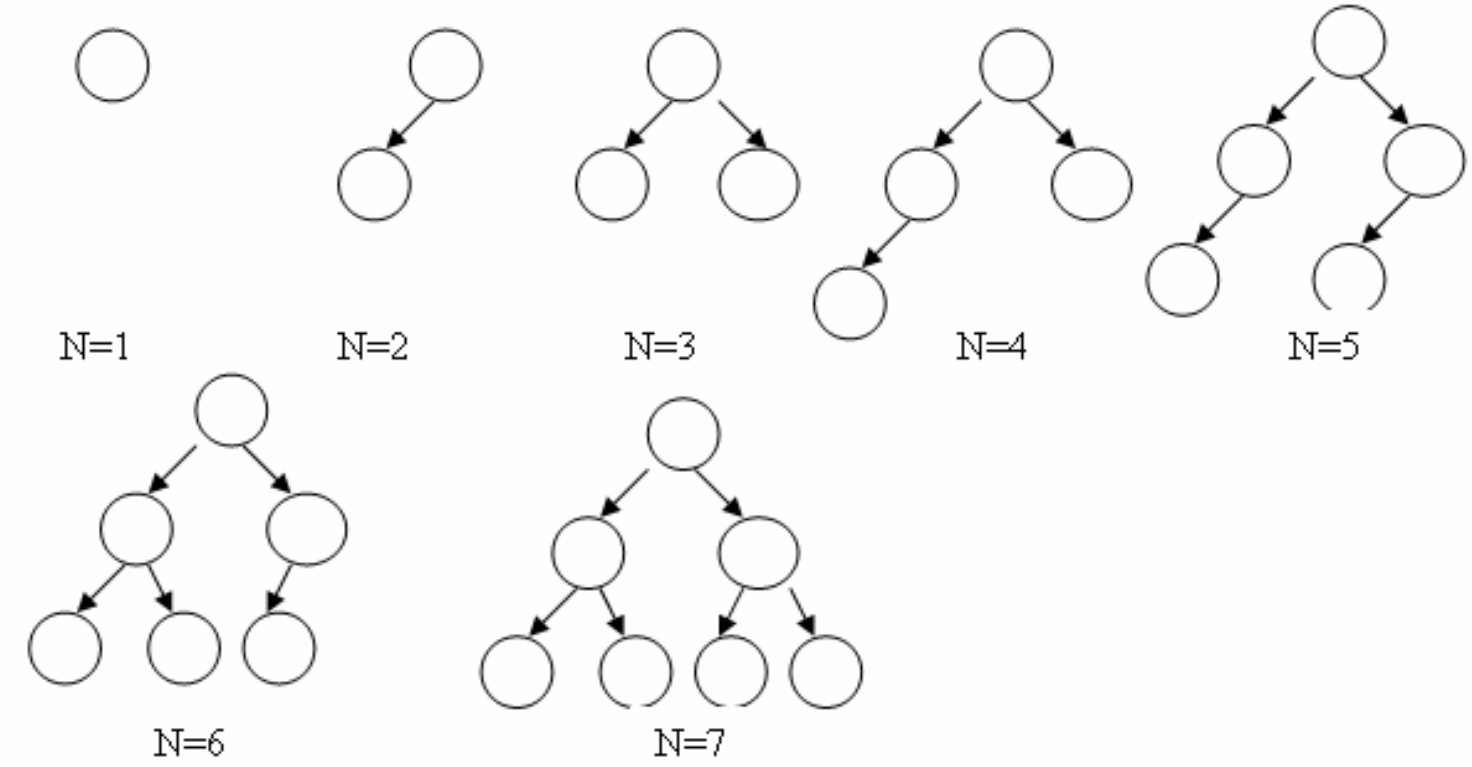
Pohon Seimbang

- Pohon seimbang (*balanced tree/B-tree*) adalah pohon dengan:
  - Perbedaan tinggi subpohon kiri dan subpohon kanan maksimum 1
  - Perbedaan banyaknya simpul subpohon kiri dan subpohon kanan maksimum 1
- Aplikasi: pengelolaan indeks dalam file system dan database system
- Yang akan dibahas adalah pohon biner seimbang (*balanced binary tree*)

Algoritma untuk membuat pohon biner seimbang dari n buah node


```
function BuildBalancedTree (n : integer) → BinTree
{ Menghasilkan sebuah balanced tree }
{ Basis: n = 0: Pohon kosong  }
{ Rekurens: n>0: partisi banyaknya node anak kiri dan kanan,
   lakukan proses yang sama  }
KAMUS LOKAL
    P : address; L, R : BinTree; X : infotype
    nL, nR : integer
ALGORITMA
    if (n = 0) then { Basis-0 }
       → Nil
    else {Rekurens }
       { bentuk akar }
       input(X) { mengisi nilai akar }
       P ← Alokasi(X)
       if (P ≠ Nil) then
          { Partisi sisa node sebagai anak kiri dan anak kanan }
          nL ← n div 2; nR ← n – nL - 1
          L ← BuildBalancedTree(nL); R ← BuildBalancedTree(nR)
          Left(P) ← L; Right(P) ← R
       → P
```

Urutan Pembentukan Pohon Biner Seimbang



Binary Search Tree - 1

- Binary Search Tree (BST)/pohon biner terurut/pohon biner pencarian adalah pohon biner yang memenuhi sifat:
  - Setiap simpul dalam BST mempunyai sebuah nilai
  - Subpohon kiri dan subpohon kanan merupakan BST
  - Jika P adalah sebuah BST:
    - semua simpul pada subpohon kiri < Akar(P)
    - semua simpul pada subpohon kanan >= Akar(P)
- Aplikasi BST: algoritma searching dan sorting tingkat lanjut

Binary Search Tree - 2

- Nilai simpul (Key) dalam BST bisa unik bisa juga tidak.
- Pada pembahasan ini semua simpul BST (Key) bernilai unik. Banyak kemunculan suatu nilai Key disimpan dalam field Count.

```
type infotype : < Key : ..., { terdefinisi }
                  Count : integer >
type Node : < Info : infotype,
              Left : BinTree,
              Right : BinTree >
type BinTree : address

{ Selektor : Jika P adalah BinTree,
  Key(P)   → akses bagian P.Info.Key
  Count(P) → akses bagian P.Info.Count
  Left(P)  → akses bagian P.Left
  Right(P) → akses bagian P.Right }
```

Insert Node dalam BST

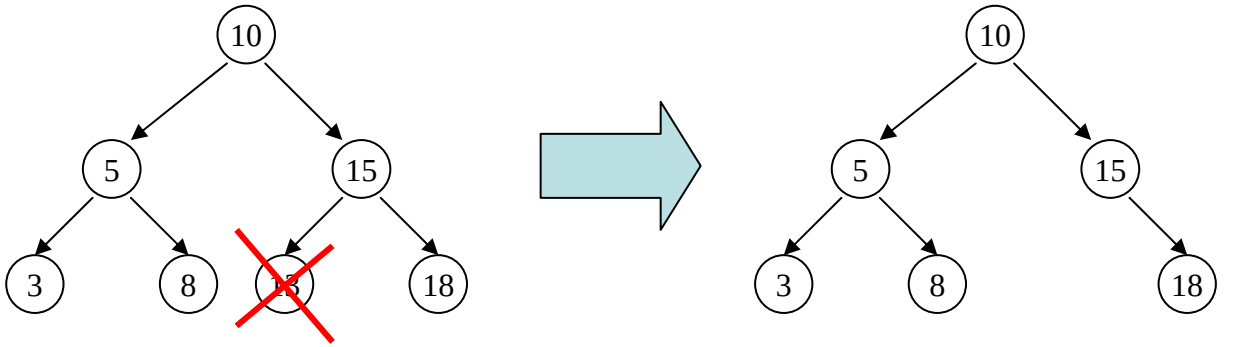
```
procedure InsSearchTree (input X : infotype,
                         input/output P : BinTree)
{ Menambahkan sebuah node X ke pohon biner pencarian P }
{ infotype terdiri dari key dan count. Key menunjukkan nilai unik,
dan Count berapa kali muncul }
{ Basis : Pohon kosong  }
{ Rekurens : Jika pohon tidak kosong, insert ke anak kiri jika
nilai < Key(P) }
{ Atau insert ke anak kanan jika nilai > Key(P) }
{ Perhatikan bahwa insert selalu menjadi daun terkiri/terkanan dari
subpohon }
{ Asumsi: Alokasi selalu berhasil }
KAMUS LOKAL

ALGORITMA
   if (IsTreeEmpty(P)) then { Basis: buat hanya akar }
      MakeTree(X,Nil,Nil,P)
   else {Rekurens }
      depend on X, Key(P)
         X.Key = Key(P) : Count(P) ← Count(P) + 1
         X.Key < Key(P) : InsSearchTree(X,Left(P))
         X.Key > Key(P) : InsSearchTree(X,Right(P))
```

Delete Simpul dalam BST - 1

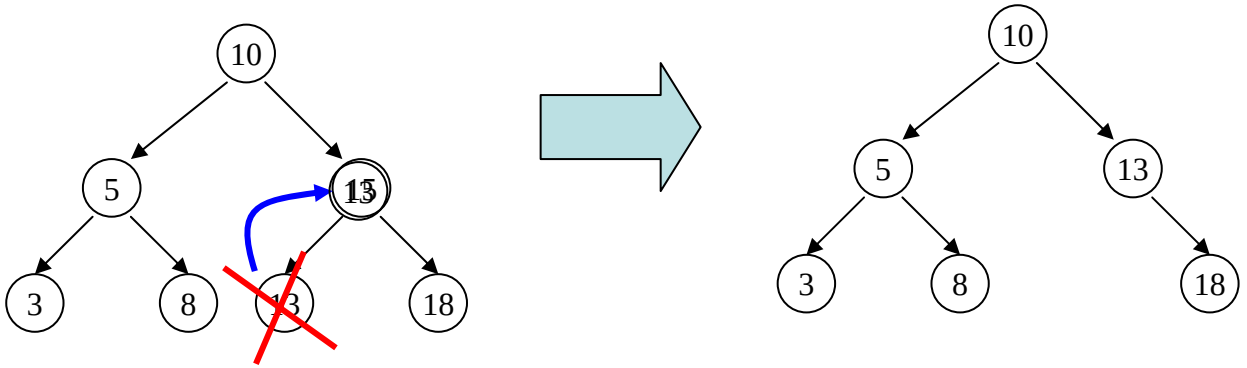

```
procedure DelBTree (input/output P : BinTree, input X : infotype)
{ Menghapus simpul bernilai Key(P) = X }
{ infotype terdiri dari key dan count. Key menunjukkan nilai
unik, dan Count berapa kali muncul }
{ Basis : ? ; Rekurens : ? }
KAMUS LOKAL
   q : address
   procedure DelNode (input/output P : BinTree)
   { … }
ALGORITMA
   depend on X, Key(P)
     X.Key < Key(P) : DelBTree(Left(P),X)
     X.Key > Key(P) : DelBTree(Right(P),X)
     X.Key = Key(P) : { Delete simpul ini }
                      q ← P
                      if Right(q) = Nil then P ← Left(q)
                      else if Left(q) = Nil then P ← Right(q)
                      else
                          DelNode(Left(q))
                      Dealokasi(q)
```

Delete Simpul dalam BST - 2

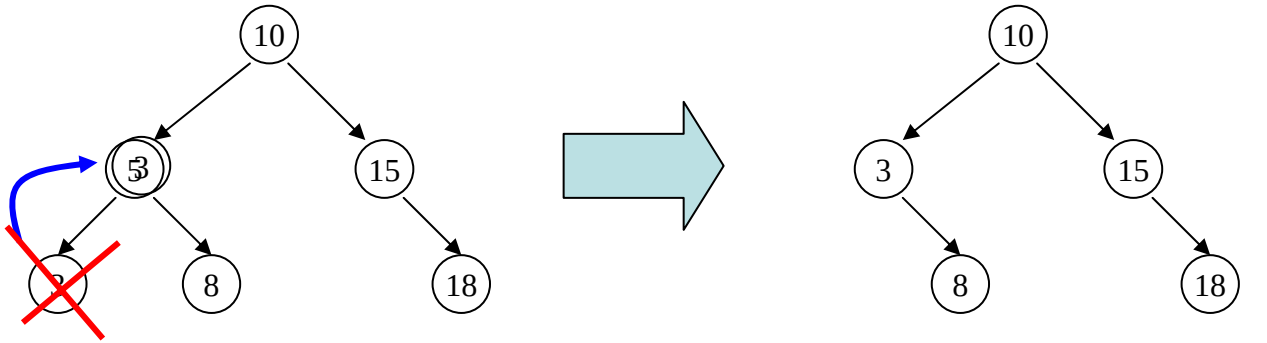
```
procedure DelNode (input/output P: BinTree)
{ I.S. P adalah pohon biner tidak kosong }
{ F.S. q berisi salinan nilai daun terkanan }
{ Proses : }
{ Memakai nilai q yang global}
{ Traversal sampai daun terkanan, copy nilai daun terkanan P,
salin nilai ke q semula }
{ q adalah anak terkiri yang akan dihapus }
KAMUS LOKAL

ALGORITMA
   depend on P
      Right(P) ≠ Nil : DelNode(Right(P))
      Right(P) = Nil : Key(q) ← Key(P)
                       Count(q) ← Count(P)
                       q ← P
                       P ← Left(P)
```



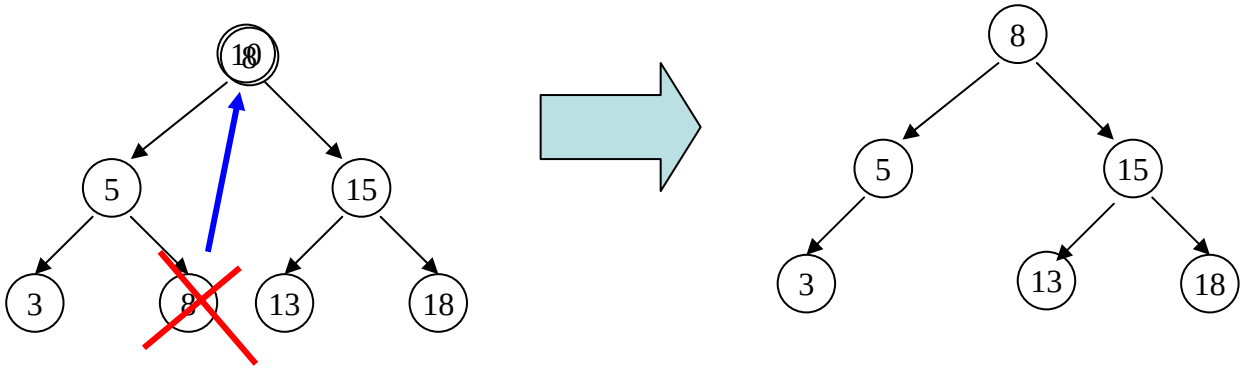
DelBTree(P,13)



DelBTree(P,15)



DelBTree(P,5)



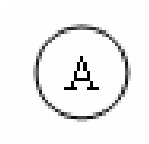
DelBTree(P,10)

Membentuk Pohon Biner dari Pita Karakter

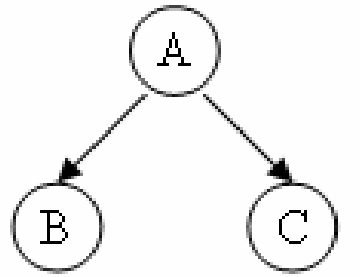
- Ekspresi pohon dalam bentuk linier (list) dapat dituliskan dalam sebuah pita karakter
- Ada 2 ide:
  - Membangun pohon secara iteratif
  - Membangun pohon secara rekursif

Contoh - 1

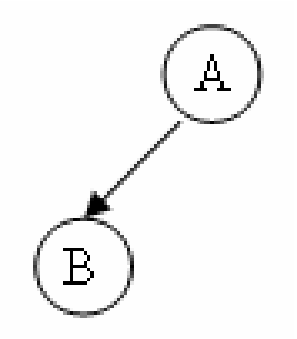
(A()())



(A(B()())(C()()))

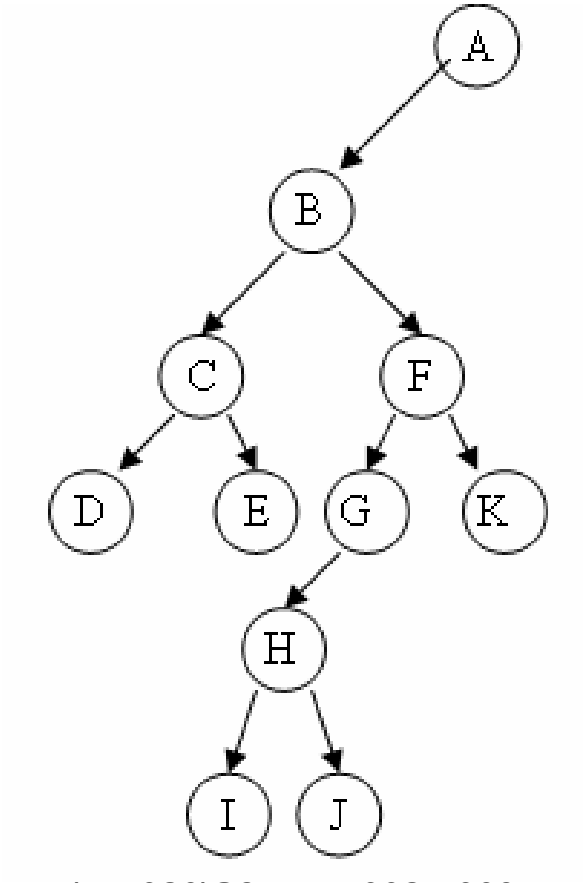


(A(B()())())



Contoh - 2

(A(B(C(D()())(E()()))(F(G(H(I()())(J()()))) (K()()) ())



Ide 1 : Membangun Pohon secara Iteratif

- Karena pembacaan pita dilakukan secara sekuensial, pembentukan pohon selalu dimulai dari akar
- Pembacaan karakter demi karakter dilakukan secara iteratif, untuk membentuk sebuah pohon, selalu dilakukan insert terhadap daun
- Struktur data memerlukan pointer ke “Bapak”, dengan demikian yang dipakai adalah:

```
type Node: < Parent : address,
             Left   : address,
             Info   : character,
             Right  : address     >
```

Ide Algoritma Membangun Pohon

- Ada tiga kelompok karakter:
  - Karakter berupa abjad, menandakan bahwa sebuah node harus dibentuk, entah sebagai anak kiri atau anak kanan.
  - Karakter berupa ‘(‘ menandakan suatu sub pohon baru.
    - Jika karakter sebelumnya adalah ‘)’ maka siap untuk melakukan insert sub pohon kanan.
    - Jika karakter sebelumnya adalah abjad, maka siap untuk melakukan insert sub pohon kiri.
  - Karakter berupa ‘)’ adalah penutup sebuah pohon, untuk kembali ke “Bapaknya”, berarti naik levelnya dan tidak melakukan apa-apa, tetapi menentukan proses karakter berikutnya.
- Tidak cukup dengan mesin karakter (hanya CC), sebab untuk memproses sebuah karakter, dibutuhkan informasi karakter sebelumnya → karena itu digunakan mesin couple (C1, CC)

Implementasi dalam Bahasa C
File : tree.h

```
#include <stdlib.h>
#include "boolean.h"
#include "mesincouple.h"

typedef char infotype;
#define Nil NULL

/*** Selektor ****/
#define Info(P) (P)->Info
#define Left(P) (P)->Left
#define Right(P) (P)->Right
#define Parent(P) (P)->Parent

/*** Type Tree ***/
typedef struct tElmtTree *address;
typedef struct tElmtTree {
   infotype Info;
   address Left;
   address Right;
   address Parent;
} ElmtTree;
typedef address Tree;
```

Implementasi dalam Bahasa C
File : tree.h

```
void Alokasi (address *P, infotype X);
/* I.S. sembarang */
/* F.S. address P dialokasi, dan bernilai tidak Nil jika
berhasil */
/* Alokasi sebuah address P */

void MakeTree(Tree *T);
/* I.S. Sembarang */
/* F.S. T terdefinisi */
/* Proses : Membaca isi pita karakter dan membangun pohon
dilakukan secara iteratif. Pita karakter mungkin kosong. */

void PrintTree (Tree T);
/* I.S. T terdefinisi */
/* F.S. T tertulis di layar */
```

Implementasi dalam Bahasa C
File : tree.c
Prosedur MakeTree
Asumsi tambahan: pitakarakter mungkin kosong

```
void MakeTree (Tree *T)
{  /* Kamus Lokal */
   address CurrParent;
   address Ptr;
   int level = 0;
   boolean InsKi;
   /* Algoritma */
   START_COUPLE();
   CurrParent = Nil;
   while (!EOP()) {
        switch (CC) {
            case '(' : level++;
                       InsKi = C1 != ')';
                       break;
            case ')' : level--;
                       if (C1 != '(') {
                          CurrParent = Parent(CurrParent);
                       }
                       break;
            default  : Alokasi (&Ptr,CC);
                       if (CurrParent != Nil) {
                          if (InsKi) { Left(CurrParent) = Ptr; }
                          else { Right(CurrParent) = Ptr; }
                       } else { *T = Ptr; }
                       Parent(Ptr) = CurrParent;
                       CurrParent = Ptr;
                       break;
        }
        ADV_COUPLE();
   }
}
```

Ide 2 : Membangun Pohon Secara Rekursif - 1

- Struktur data yang digunakan adalah tree biasa (tidak memerlukan pointer ke Bapak)

```
typedef struct tElmtTree *address;
typedef struct tElmtTree {
   infotype Info;
   address Left;
   address Right;
} ElmtTree;
typedef address Tree;
```

- Hanya memerlukan modul **mesin karakter** untuk membaca pita karakter

```
void BuildTree(Tree *T)
/* Dipakai jika input dari pita karakter */
/* I.S.: Sembarang */
/* F.S.: T terdefinisi */
/* Proses : Membaca isi pita karakter dan membangun pohon secara
            rekursif */
```

Ide 2 : Membangun Pohon secara Rekursif - 2

```
void BuildTree(Tree *T)
/* Dipakai jika input dari pita karakter */
/* I.S. Sembarang */
/* F.S. T terdefinisi */
/* Proses : Membaca isi pita karakter dan membangun pohon secara
rekursif, hanya membutuhkan mesin karakter */
{   /* Kamus Lokal */

    /* Algoritma */
    ADV();              /* advance */
    if (CC==')')        /* Basis : pohon kosong */
        (*T)=Nil;
    else {              /* Rekurens */
        Alokasi(T,CC);
        ADV();          /* advance */
        BuildTree(&(Left(*T)));
        BuildTree(&(Right(*T)));
    }
    ADV();              /* advance */
}
```

Ide 2 : Membangun Pohon secara Rekursif - 3

- Contoh pemanggilan di program utama:

```
#include "tree.h"

int main () {
    /* KAMUS */
    Tree T;

    /* ALGORITMA */
    START();
    BuildTree(&T);
    PrintTree(T); /* mencetak pohon */
    return 0;
}
```

Membangun Pohon dari String - 1

- Menggunakan ide pembangunan pohon dari pita karakter secara rekursif
- Struktur data yang digunakan adalah struktur data pohon biasa (tidak perlu pointer ke Bapak)

```
void BuildTreeFromString (Tree *T, char *st, int *idx)
/* Input dari string st */
/* I.S. Sembarang */
/* F.S. T terdefinisi */
/* Proses : Membaca string st dan membangun pohon secara rekursif */
```

Membangun Pohon dari String - 2

```
void BuildTreeFromString (Tree *T, char *st, int *idx)
/* Input dari string st */
/* I.S. Sembarang */
/* F.S. T terdefinisi */
/* Proses : Membaca string st dan membangun pohon secara rekursif
*/
{  /* Kamus Lokal */
   /* Algoritma */
   (*idx)++;                              /* advance */
   if (st[*idx]==')')  /* Basis : pohon kosong */
      (*T)=Nil;
   else {              /* Rekurens */
      Alokasi(T,st[*idx]);
      (*idx)++;                            /* advance */
      BuildTreeFromString(&Left(*T),st,idx);
      BuildTreeFromString(&Right(*T),st,idx);
   }
   (*idx)++;                              /* advance */
}
```

Membangun Pohon dari String - 3

- Contoh pemanggilan di program utama:

```
#include "tree.h"

int main () {
    /* KAMUS */
    Tree T;
    char *S = "(A()())";
    int idx = 0;

    /* ALGORITMA */
    BuildTreeFromString(&T,S,&idx);
    PrintTree(T); /* mencetak pohon */
    return 0;
}
```

PR

- Melanjutkan modul P-15. ADT Pohon Biner untuk fungsi/prosedur yang terkait:
  - Pohon seimbang
  - Pohon biner terurut
  - Pembangunan pohon dari pita karakter/string