POHON

Pendahuluan
Struktur pohon adalah struktur yang penting dalam bidang informatika, yang memungkinkan kita untuk :

- mengorganisasi informasi berdasarkan sutau struktur "lojik"
- memungkinkan cara akses yang bermacam-macam terhadap suatu elemen

Contoh persoalan yang tepat untuk direpresentasi sebagai pohon:

- pohon keputusan,
- pohon keluarga dan klasifikasi dalam botani,
- pohon sintaks dan pohon ekspresi aritmatika
- pohon "dekomposisi" bab dari sebuah buku
- pohon "menu" dari suatu aplikasi komputer

Definisi rekurens dari pohon:
Sebuah POHON adalah himpunan terbatas tidak kosong, dengan elemen yang dibedakan sebagai berikut :

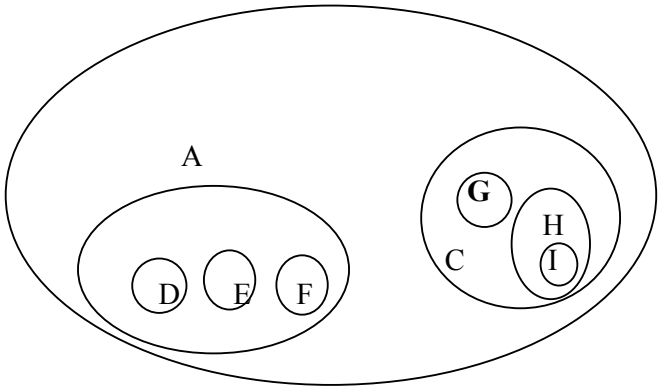
- sebuah elemen dibedakan dari yang lain, yang disebut sebagai AKAR dari pohon
- elemen yang lain (jika masih ada) dibagi-bagi menjadi beberapa sub himpunan yang disjoint, dan masing-masing sub himpunan tersebut adalah POHON yang disebut sebagai SUB POHON dari POHON yang dimaksud.

Contoh :
Sebuah buku dipandang sebagai pohon. Judul buku adalah AKAR. Buku dibagi menjadi bab-bab. Masing-masing bab adalah sub pohon yang juda mengandung JUDUL sebagai AKAR dari bab tersebut. Bab dibagi menjadi sub bab yang juga diberi judul. Sub bab adalah pohon dengan judul sub bab sebagai akar. Daftar Isi buku dengan penulisan yang di-identasi mencerminkan struktur pohon dari buku tersebut.

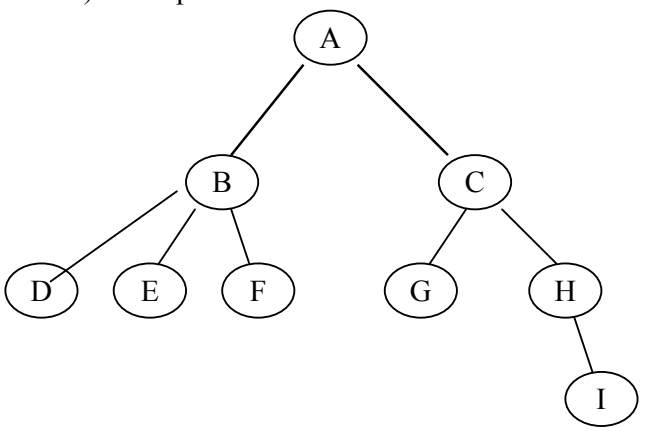
Catatan:
Suffiks (akhiran) n-airy menunjukkan bahwa sub pohon bervariasi
semua elemen dari pohon adalah akar dari sub pohon, yang sekaligus menunjukkan pohon tsb
pada definisi di atas, tidak ada urutan sub pohon, namun jika logika dari persoalan mengharuskan suatu strukturasi seperti halnya pada buku, maka dikatakan bahwa pohon berarah

CARA PENULISAN POHON:
Beberapa ilustrasi representasi berikut ini yang diambil dari [3] merepresentasikan pohon yang sama

a) Himpunan yang saling melingkupi



b) Graph



c) Indentasi

A
B
D
E
F
C
G
H
I

d) Bentuk linier :

Prefix : (A (B(D(),E(),F()), C(G(),H(I()))))), atau
(A(B(D)(E)(F))(C(G)(H(I))))
Posfix : ((D,E,F)B,(G,(I) H) C))

BEBERAPA ISTILAH

HUTAN *(forest)*
Definisi : hutan adalah *sequence* (list) dari pohon)

Jika kita mempunyai sebuah hutan, maka kita dapat menambahkan sebuah akar fiktif pada hutan tersebut dan hutan tersebut menjadi list dari sub pohon. Demikian pula sebaliknya: jika diberikan sebuah pohon dan kita membuang akarnya, maka akan didapatkan sebuah hutan.

SIMPUL (*node, elemen*) : adalah elemen dari pohon yang memungkinkan akses pada sub pohon dimana simpul tersebut berfungsi sebagai AKAR.

CABANG (*path*): hubungan antara akar dengan sub pohon
Contoh : pada gambar (B) di atas

Maka A dihubungkan dengan B dan C, untuk menunjukkan bahwa AKAR A dan kedua himpunan {B,D,E,F} dan {C,G,H,I} masing-masing adalah pohon dengan akar B dan C.

AYAH (*father)* : Akar dari sebuah pohon adalah AYAH dari sub pohon.

ANAK (*child)* : ANAK dari sebuah AKAR adalah sub pohon.

SAUDARA (*sibling)* : adalah simpul-simpul yang mempunyai AYAH yang sama.

DAUN (*leaf)* adalah simpul terminal dari pohon. Semua simpul selain daun adalah simpul BUKAN-TERMINAL.

JALAN (***path***) adalah suatu urutan tertentu dari CABANG
DERAJAT sebuah pohon adalah banyaknya anak dari pohon tersebut. Sebuah simpul berderajat N disebut sebagai pohon N-aire. Pada pohon biner, derajat dari sebuah simpul mungkin 0-aire (daun), 1 -aire/uner atau 2-aire/biner.

TINGKAT (***Level***) pohon adalah panjangnya jalan dari AKAR sampai dengan simpul yang bersangkutan. Sebagai perjanjian, panjang dari jalan adalah banyaknya simpul yang dikandung pada jalan tersebut. Akar mempunyai tingkat sama dengan 1. Dua buah simpul disebut sebagai saudara jika mempunyai tingkat yang sama dalam suatu pohon.

KEDALAMAN (*depth)* sebuah pohon adalah nilai maksimum dari tingkat simpul yang ada pada pohon tersebut. Kedalaman adalah panjang maksimum jalan dari akar menuju ke sebuah daun.

LEBAR (*breadth)* sebuah pohon adalah maksimum banyaknya simpul yang ada pada suatu tingkat.

Catatan:
Diberikan sebuah pohon biner dengan N elemen. Jika :
- b adalah banyaknya simpul biner
- u adalah banyaknya simpul uner
- d adalah banyaknya daun

Maka akan selalu berlaku:

$N = b + u + d$

$n-1 = 2\ b + u$

sehingga

$b = d - 1$

Representasi ponon n-airy (Pohon N-ner) : adalah dengan **list of list**

Pohon N-aire

Pohon N-aire adalah pohon yang pada setiap level anaknya boleh berbeda-beda jumlahnya, dan anaknya tersebut adaalah pohon N-aire

Definisi rekursif

- Basis-1 : pohon yang hanya terdiri dari akar adalah pohon N-aire
- Rekurens : Sebuah pohon N-aire terdiri dari akar dan sisanya ("anak-anak"nya) adalah list pohon N-aier.

Pada definisi rekrusif tersebut tidak dicakup pohon kosong, karena pohon N-aire tidak pernah kosong

TYPE POHON-N-AIRE (tidak mungkin kosong)

DEFINISI DAN SPESIFIKASI TYPE

type Elemen : { tergantung type node }

type PohonN-ner : < A : Elemen, PN : PohonN-ner > { notasiPrefix }, atau

type PohonN-ner : < PN : PohonN-ner, A : Elemen > { notasi postfix }

{Pohon N-ner terdiri dari Akar yang berupa elemen dan list dari pohon N-aire yang menjadi anaknya List anak mungkin kosong, tapi pohon N-ner tidak pernah kosong, karena minimal mempunyai sebuah elemen sebagai akar pohon}

DEFINISI DAN SPESIFIKASI SELEKTOR

Akar : PohonN-ner tidak kosong → Elemen

{ Akar(P) adalah Akar dari P. Jika P adalah (A,PN) = Akar(P) adalah A }

Anak : PohonN-ner tidak kosong → list of PohonN-ner

{ Anak(P) adalah list of pohon N-ner yang merupakan anak-anak (sub phon) dari P. Jika P adalah (A, PN) = Anak (P) adalah PN }

DEFINISI DAN SPESIFIKASI KONSTRUKTOR

{ Perhatikanlah bahwa konstruktor pohon N-ner dengan basis pohon kosong dituliskan sebagai

a.Prefix : (A,P,N)

b. Posfix : (PN,A) }

DEFINISI DAN SPESIFIKASI PREDIKAT

IsTreeNEmpty : PohonN-ner → boolean

{IsTreeNEmpty(PN) true jika PN kosong : () }

IsOneElmt : PohonN-ner → boolean

{IsOneElmt(PN) true jika PN hanya terdiri dari Akar }

DEFINISI DAN SPESIFIKASI PREDIKAT LAIN

NbNElmt : PohonN-ner → integer ≥ 0

{NbNElmt(P) memberikan banyaknya node dari pohon P :
Basis 1: NbNElmt ((A)\) = 1
Rekurens : NbNElmt ((A,PN)) = 1 + NbELmt(PN) }

NbNDaun : PohonN-ner → integer ≥ 0

{NbNDaun (P) memberikan banyaknya daun dari pohon P :
Basis 1: NbNDaun (A) = 1
Rekurens : NbDaun ((A,PN)) = NbNDaun(PN)

Realisasi pohon ini menjadi list of list tidak dibuat, dan sengaja diberikan sebagai latihan .

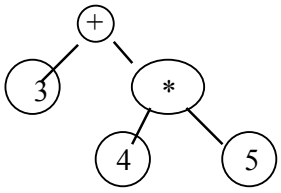
Pohon Biner

Definisi :

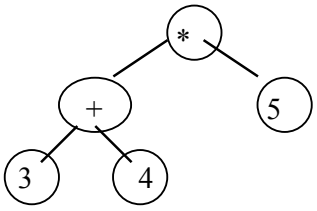
sebuah pohon biner adalah himpunan terbatas yang
- mungkin kosong, atau
- terdiri dari sebuah simpul yang disebut akar
  dan dua buah himpunan lain yang *disjoint* yang merupakan **pohon biner**, yang disebut sebagai sub pohon kiri dan sub pohon kanan dari pohon biner tersebut

Perhatikanlah perbedaan pohon biner dengan pohon biasa : pohon biner mungkin kosong, sedangkan pohon n-aire tidak mungkin kosong.

Contoh pohon ekspresi aritmatika

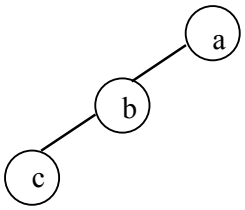


3+ (4*5)

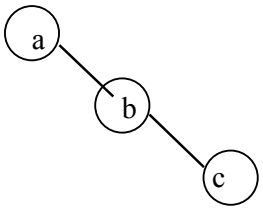


(3+4) * 5

Karena adanya arti bagi sub pohon kiri dan sub pohon kanan, maka dua buah pohon biner sebagai berikut berbeda (pohon berikut disebut pohon condong/skewed tree)

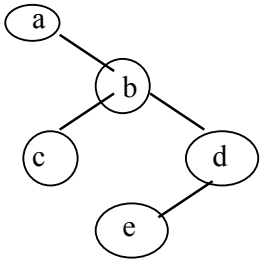


Pohon biner condong kiri



Pohon biner condong kanan

Sub pohon ditunjukkan dengan penulisan ( )



Notasi prefix :

(a ( ), (b (c () ())(d (e) ()))), atau
(a() (b (c) (d (e) ())))

Definisi rekursif pohon biner basis-0

- **Basis** : pohon biner kosong adalah pohon biner
- **Rekurens** : Pohon biner yang tidak kosong, terdiri dari sebuah node yang disebut akar, dan sub pohon kiri dan sub pohon kanan sebagai anak-anaknya yang juga merupakan pohon biner.

Jika pada list hanya ada dua cara melakukan konstruksi/seleksi yaitu pertama atau terakhir (perhatikan kata terdiri dari …), maka pada pohon biner tiga alternatif berikut dapat dipilih yaitu infix, prefix dan postfix. Pemilihan salah satu cara untuk implementasi disesuaikan dengan bahasanya. Contohnya karena dalam LISP ekspresi ditulis prefix, maka akan lebih mudah kalau dipilih secara prefix.

TYPE POHON BINER: Model -0, dengan basis pohon kosong

DEFINISI DAN SPESIFIKASI TYPE

type Elemen : { tergantung type node }

type PohonBiner : <L : PohonBiner, A : Elemen, R : PohonBiner > {notasi Infix}, atau

type PohonBiner : < A : Elemen, L : PohonBiner, R : PohonBiner > {notasi prefix}, atau

type PohonBiner : <L : PohonBiner, R : PohonBiner, A : Elemen > {notasi postfix }

{Pohon Biner terdiri dari Akar yang berupa elemen, L dan R adalah Pohon biner yang merupakan subPohon kiri dan subpohon kanan }

DEFINISI DAN SPESIFIKASI SELEKTOR

Akar : PohonBiner tidak kosong → Elemen

{ Akar(P) adalah Akar dari P. Jika P adalah /L,A,R\ = Akar(P) adalah A }

Left : PohonBiner tidak kosong → PohonBiner

{ Left(P) adalah sub pohon kiri dari P. Jika P adalah /L,A,R\ = Left (P) adalah L }

Right : PohonBiner tidak kosong → PohonBiner

{Right(P) adalah sub pohon kanan dari P. Jika P adalah /L,A,R\ = Right (P) adalah R }

DEFINISI DAN SPESIFIKASI KONSTRUKTOR

{Perhatikanlah bahwa konstruktor pohon biner dengan basis pohon kosong dituliskan sebagai

a. Infix : /L A R

b. Prefix : /A L R

c. Posfix : /L R A\ }

DEFINISI DAN SPESIFIKASI PREDIKAT

IsTreeEmpty : PohonBiner → boolean

{IsTreeEmpty (P) true jika P adalah Pohon biner kosong : (/ \) }

DEFINISI DAN SPESIFIKASI PREDIKAT LAIN

NbElmt : PohonBiner → integer ≥ 0

{NbElmt(P) memberikan Banyaknya elemen dari pohon P :
Basis : NbElmt (//\ \) = 0
Rekurens : NbElmt (/L,A,R\) = NbElmt(L) + 1 + NbELmt(R) }

NbDaun : PohonBiner → integer ≥ 0

{ definisi : Pohon kosong berdaun 0 }
{NbDaun (P) memberikan Banyaknya daun dari pohon P :
Basis-1 : NbDaun (//\ \) = 0
Rekurens :
***NbDaun1** (P)*

RepPrefix: PohonBiner → list of element

{RepPrefix (P) memberikan representasi linier (dalam bentuk list), dengan urutan elemen list sesuai dengan urutan penulisan pohon secara prefix :
Basis : RepPrefix (//\ \) = []
Rekurens :RepPrefix (/L,A,R\) = [A] o RepPrefix(L) o RepPrefix (R) }

REALISASI

```
NbElmt (P) : {boleh model basis-0 }
   if IsTreeEmpty?(P) then {Basis 0} 0
   else {Rekurens } NbElmt(Left(P) + 1 + NbElmt(Right(P)


NbDaun (P) :
 if IsEMpty?(P) then 0
 else {Pohon tidak kosong:minimal mempunyai satu akar, sekaligus daun}
   { aplikasi terhadap Jumlah Daun untuk Basis-1 }
     NbDaun1 (P)


RepPrefix (P) :
   if IsTreeEmpty(P) then {Basis 0} []
   else {Rekurens }
     KonsoL(KonsoL(Akar(P), RepPrefix(Left(P)), RepPrefix(Right(P)))
```

Definisi rekursif pohon biner basis-1

- **Basis** : pohon biner yang hanya terdiri dari akar
- **Rekurens** : Pohon biner yang tidak kosong, terdiri dari sebuah node yang disebut akar, dan sub pohon kiri dan sub pohon kanan sebagai anak-anaknya yang juga merupakan pohon biner tidak kosong

TYPE POHON BINER : Model-1: pohon minimal mempunyai satu elemen

DEFINISI DAN SPESIFIKASI TYPE

type Elemen : { tergantung type node }

typ} PohonBiner : <L : PohonBiner, A : Elemen, R : PohonBiner > {notasi Infix}, atau

type PohonBiner : < A : Elemen, L : PohonBiner, R : PohonBiner > {notasi prefix }, atau

type PohonBiner : <L : PohonBiner, R : PohonBiner, A : Elemen > {notasi postfix }

{Pohon Biner terdiri dari Akar yang berupa elemen, L dan R adalah Pohon biner yang merupakan subPOhon kiri dan subpohon kanan }

DEFINISI DAN SPESIFIKASI SELEKTOR

Akar : PohonBiner tidak kosong → Elemen

{ Akar(P) adalah Akar dari P. Jika P adalah //L A R\\ = Akar(P) adalah A }

Left : PohonBiner tidak kosong → PohonBiner

{ Left(P) adalah sub pohon kiri dari P. Jika P adalah //L A R\ \, Left (P) adalah L }

Right : PohonBiner tidak kosong → PohonBiner

{Right(P) adalah sub pohon kanan dari P. Jika P adalah //L A R\\,Right (P) adalah R}

DEFINISI DAN SPESIFIKASI KONSTRUKTOR

{Perhatikanlah bahwa konstruktor pohon biner dengan basis pohon kosong dituliskan sebagai

a. Infix : //L A R

b. Prefix : //A L R

c. Posfix : //L R A

atau bahkan notasi lain yang dipilih}

DEFINISI DAN SPESIFIKASI PREDIKAT

IsTreeEmpty : PohonBiner → boolean

{IsTreeEmpty(P) true jika P kosong : (// \\) }

IsOneElmt : PohonBiner → boolean

{IsOneElement(P) true jika P hanya mempunyai satu elemen, yaitu akar (// A \\\) }

IsUnerLeft : PohonBiner → boolean

{IsUnerLeft(P) true jika P hanya mengandung sub pohon kiri tidak kosong: (//L A \\) }

IsUnerRight : PohonBiner → boolean

{IsUnerRight (P) true jika P hanya mengandung sub pohon kanan tidak kosong: (//A R\\) }

IsBiner : PohonBiner tidak kosong → boolean

{IsBiner(P) true jika P mengandung sub pohon kiri dan sub pohon kanan : (//L A R\\) }

IsExistLeft : PohonBiner tidak kosong → boolean

{IsExistLeft (P) true jika P mengandung sub pohon kiri }

IsExistRight : PohonBiner tidak kosong → boolean

{ExistRight(P) true jika P mengandung sub pohon kanan }

DEFINISI DAN SPESIFIKASI PREDIKAT LAIN

NbElmt : PohonBiner → integer ≥ 0

{NbElmt(P) memberikan Banyaknya elemen dari pohon P :
Basis : NbElmt (//A\ \) = 1
Rekurens : NbElmt (//L,A,R\\ ) = NbElmt(L) + 1 + NbELmt(R)
NbElmt (//L,A,\ ) = NbElmt(L) + 1
NbElmt (//A,R\\ ) = 1 + NbELmt(R) }

NbDaun1 : PohonBiner → integer ≥ 1

{ Prekondisi : Pohon P tidak kosong }

{NbDaun (P) memberikan Banyaknya daun dari pohon P :
Basis : NbDaun1 (//A\ \) = 1
Rekurens : NbDaun1 (//L,A,R\\) = NbDaun1 (L) + NbDaun1(R)
NbDaun1 (//L,A,\\) = NbDaun1 (L)
NbDaun1 (//A,R\\) = NbDaun1 (R)

RepPrefix: PohonBiner → list of element

{RepPrefix (P) memberikan representasi linier (dalam bentuk list), dengan urutan elemen list sesuai dengan urutan penulisan pohon secara prefix :
Basis : RepPrefix (//A\ \) = [A]
Rekurens : RepPrefix (//L,A,R\\) = [A] o RepPrefix(L) o RepPrefix (R)
RepPrefix (//L,A\\) = [A] o RepPrefix(L)
RepPrefix (//A,R\\) = [A] o RepPrefix (R)
}

REALISASI

```
NbElmt (P) : {P tidak kosong }
   if IsOneElmt(P) then 1
   else depend on P
            IsBiner(P) : NbElmt(Left(P) + 1 + NbElmt(Right(P)
            IsUnerLeft(P) : NbElmt(Left(P) + 1
```

```
            IsUnerRight(P) : 1 + NbElmt(Right(P)

NbDaun (P) :
   if 1Element?(P) then {Basis}
        1
   else {Rekurens }
         depend on P
            IsBiner(P) : NbDaun1(Left(P)) +  NbDaun1(Right(P))
            IsUnerLeft(P) : NbDaun1(Left(P))
            IsUnerRight(P) : NbDaun1(Right(P))

RepPrefix (P) :
   if 1Elmt?(P) then [Akar(P)]
   else depend on P
            IsBiner(P):KonsoL(KonsoL(Akar(P),RepPrefix(Left(P)),
                      RepPrefix(Right(P)))
            IsUnerLeft(P) : KonsoL(Akar(P),RepPrefix(Left(P))
            IsUnerRight(P) : KonsoL(Akar(P),RepPrefix(Right(P))
```

Latihan soal :

1. Pelajarilah apakah semua predikat pada model 1 berlaku untuk model 0
2. Buatlah spesifikasi dan definisi dari sebuah fungsi yang memeriksa apakah suatu nilai X ada sebagai simpul sebuah pohon
3. Buatlah spesifikasi dan definisi dari sebuah fungsi yang memeriksa apakah suatu pohon biner adalah sebuah pohon biner condong kiri
4. Buatlah spesifikasi dan definisi dari sebuah fungsi yang memeriksa apakah suatupohon biner adalah sebuah pohon biner condong kanan
5. Buatlah spesifikasi dan definisi dari sebuah fungsi yang menghitung banyaknya operator uner pada sebuah pohon ekspresi aritmatika infix
6. Buatlah spesifikasi dan definisi dari sebuah fungsi yang memeriksa banyaknya simpul yang ada pada level n
7. Latihan :

Buatlah realisasi dari spesifikasi fungsional terhadap pohon biner sebagai berikut

DEFINISI DAN SPESIFIKASI PREDIKAT LAIN

IsMember : PohonBiner, elemen → boolean

{IsMember(P,X) Mengirimkan true jika ada node dari P yg bernilai X }

{ fungsi lain }

IsSkewLeft: PohonBiner → boolean

{ IsSkewLeft(P) Mengirimkan true jika P adalah pohon condong kiri }

IsSkewRight : PohonBiner → boolean

{ IsSkewRight(P) Mengirimkan true jika P adalah pohon condong kiri }

LevelOfX: PohonBiner, elemen → integer

{ LevelOfX(P,X) Mengirimkan level dari node X yang merupakan salah satu simpul dari pohon biner P }

{Operasi lain }

AddDaunTerkiri : PohonBiner, elemen → PohonBiner

{AddDaunTerkiri(P,X): mengirimkan Pohon Biner P yang telah bertambah simpulnya, dengan X sebagai simpul daun terkiri }

AddDaun : PohonBiner tidak kosong, node,node, boolean → PohonBiner

{AddDaun (P,X,Y,Kiri) : P bertambah simpulnya, dengan Y sebagai anak kiri X (jika Kiri), atau sebagai anak Kanan X (jika not Kiri)
{ Prekondisi : X adalah salah satu daun Pohon Biner P }

DelDaunTerkiri: PohonBiner tidak kosong, → <PohonBiner,elemen >

{DelDaunTerkiri(P) menghasilkan sebuah pohon yang dihapus daun terkirinya, dengan X adalah info yang semula disimpan pada daun terkiri yang dihapus }

DelDaun : PohonBiner tidak kosong, elemen → PohonBiner

{ DelDaun(P,X) dengan X adalah salah satu daun , menghasilkan sebuah pohon tanpa X yang semula adalah daun dari P}

MakeListDaun : PohonBiner →list Of Node

{MakeListDaun(P) : }
{Jika P adalah pohon kosong, maka menghasilkan list kosong.
{Jika P bukan pohon kosong: menghasilkan list yang elemennya adalah semua daun pohon P}

MakeListPreOrder : PohonBiner) →list Of Node

{MakeListPreOrder(P) : }
{Jika P adalah pohon kosong, maka menghasilkan list kosong. }
{Jika P bukan pohon kosong: menghasilkan list yang elemennya adalah semua node pohon P dengan urutan Preorder}

MakeListPostOrder : PohonBiner →list Of Node

{MakeListPostOrder(P) : }
{Jika P adalah pohon kosong, maka menghasilkan list kosong. }
{Jika P bukan pohon kosong: menghasilkan list yang elemennya adalah semua node pohon P dengan urutan PostOrder}

MakeListInOrder : PohonBiner →list Of Node

{MakeListInOrder(P) : }
{Jika P adalah pohon kosong, maka menghasilkan list kosong. }
{Jika P bukan pohon kosong: menghasilkan list yang elemennya adalah semua node pohon P dengan urutan InOrder}

MakeListLevel : PohonBiner, integer →list Of Node

{MakeListLevel(P,N) : }
{Jika P adalah pohon kosong, maka menghasilkan list kosong. }
{Jika P bukan pohon kosong: menghasilkan list yang elemennya adalah semua node pohon P yang levelnya=N}

Ada pohon biner yang mempunyai sifat-sifat khusus, misalnya pohon biner pencarian (binary search tree) dan pohon seimbang. Selain semua operator dan fungsi yang berlaku untuk pohon biner, ada operator lain yang didefinisikan.

Binary Search Tree

Definisi Binary Search Tree dengan key yang unik : Jika P = /L A R\ adalah sebuah binary tree, maka:

- semua key dari node yang merupakan anak kiri P nilainya lebih kecil dari A, dan
- semua key dari node yang merupakan anak kanan P nilainya lebih besar dari A,

Definisi dan spesifikasi operasi terhadap binary search tree diberikan sebagai berikut. Realisasi nya harus dibuat sebagai latihan.

BSearchX : BinSearchTree, elemen → boolean

{ BsearchX(P,X) Mengirimkan true jika ada node dari Pohon Binary Search Tree P yang bernilai X, mengirimkan false jika tidak ada}

AddX: BinSearchTree, elemen → PohonBiner

{ AddX(P,X) Menghasilkan sebuah pohon Binary Search Tree P dengan tambahan simpul X. Belum ada simpul P yang bernilai X }

MakeBinSearchTree: list of elemen → PohonBiner

{ MakeBinSearchTree(Ls) Menghasilkan sebuah pohon Binary Search Tree P yang elemennya berasal dari elemen list Ls yang dijamin unik. }

DelBtree: BinSearchTree tidak kosong, elemen → PohonBiner

{DelBTree(P,X) menghasilkan sebuah pohon binary search P tanpa node yang bernilai X. X pasti ada sebagai salah satu node Binary Search Tree. Menghasilkan Binary SearchTree yang "kosong" jika P hanya terdiri dari X }

Pohon Seimbang (*balanced tree*)

Definisi Pohon seimbang:

- Pohon seimbang tingginya: perbedaan tinggi sub pohon kiri dengan sub pohon kanan maksimum 1
- Pohon seimbang banyaknya simpul: perbedaan banyaknya simpul sub pohon kiri dengan sub pohon kanan maksimum 1

Buatlah realisasi dari definisi dan spesifikasi sebagai berikut sebagai latihan

BuildBalanceTree: list of node, integer → BinBalTree *{ Meghasilkan sebuah balance tree dengan n node, nilai setiap node yang berasal dari list }*

EKSPRESI LAMBDA

Sampai dengan bab ini, pada definisi fungsi, **domain** suatu fungsi yang pernah dibahas hanyalah "type", yaitu merupakan type dasar atau type bentukan. Hasil dari fungsi (***range***) juga merupakan suatu nilai dengan type tertentu. Dengan definisi semacam ini, maka pada spesifikasi, nama parameter adalah suatu nama yang mewakili suatu nilai bertype tertentu.

Fungsi sebagai domain dari fungsi (parameter)

Pada kasus tertentu, dibutuhkan nama fungsi sebagai parameter (artinya domain suatu fungsi adalah fungsi), yang pada saat aplikasi baru akan ditentukan fungsi yang mana. Dalam hal ini harus ada mekanisme yang menampung definisi dan spesifikasi, yang asosiasinya baru ditentukan pada saat aplikasi. Ekspresi lambda memungkinkan hal ini terjadi. Suatu fungsi dapat "di-passing" sebagai parameter pada saat aplikasi melalui ekspresi lambda. Akibat dari aplikasi dengan ekspresi lambda, definisi dan spesifikasi "menghilang". Untuk itu, dalam notasi fungsional, sebelum mendefinisikan ekspresi lambda, definisi, spesifikasi dan realisasi fungsi dituliskan dengan nama fungsi. Baru pada tahap translasi ke bahasa pemrograman semacam LISP, aplikasi fungsi akan langsung menggunakan ekspresi lambda.

Selain "penangguhan" fungsi pada saat eksekusi, konsep ekspresi lambda dibutuhkan untuk mengeneralisasi fungsi.
Contohnya :

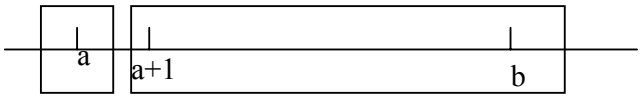
- Untuk menghasilkan sebagian elemen list dari sebuah list dengan kriteria tertentu. akan sangat praktis jika sebagai domain adalah fungsi "Filter", yang nantinya akan melakukan "filtering/pelolosan" elemen ke list hasil
- Jika kita masih belum tahu akan menentukan maksimum atau minimum, akan sangat praktis kalau didefinisikan suatu fungsi "Ekstrim" yang nantinya akan diaplikasi menjadi "Maksimum" atau "Minimum"

Beberapa persoalan matematik, menghasilkan fungsi sebagai hasil dari komputasi fungsi. Misalnya derivasi suatu fungsi polinomial akan menghasilkan suatu fungsi polinomial berderajat satu kurang dari fungsi asal. Untuk memenuhi kebutuhan ini, notasi fungsional diperluas sehingga range dari sebuah fungsi akan menghasilkan fungsi.
Translasi konsep ini ke bahasa pemrograman membutuhkan mekanisme yang sangat spesifik bahasa. Bahkan hampir tidak ada bahasa fungsional yang mampu menterjemahkan konsep ini secara satu ke satu tanpa melalui mekanisme yang tersedia.

Jadi, konsep yang dicakup dalam bab ini adalah fungsi sebagai parameter fungsi, atau bahkan sebagai hasil dari fungsi

Deskripsi persoalan : untuk suatu kebutuhan melakukan penjumlaha deret yang "mirip", mula-mula didefinisikan tiga buah fungsi yang terpisah. Semua jumlah disebut sebagai "Sigma" namun nilai yang akan dijumlahkan yang merupakan fungsi I anggota interval berbeda-beda.
Kemudian, karena kemiripan rumusnya, ingin dibentuk sebuah fungsi yang dapat mewakili ketiga fungsi tersebut. Tahapan-tahapannya diberikan lewat contoh sebagai berikut:

Perhatikan tiga buah fungsi SigI, SigI3 dan SP8 sebagai berikut :
Suatu interval akan didefinisikan secara rekurens sebagai berikut :
Basis: interval kosong artinya nilai a > b, yaitu tidak ada lagi daerah yang merupakan definisi interval
Rekurens : akan diberikan ilustrasi analisa rekurens terhadap inerval dianalogikan terhadap list sebagai berikut :



DEFINISI DAN SPESIFIKASI

$$SigI = \sum_{i=a}^{b} i$$

{SigI(a,b) adalah fungsi untuk menghitung Sigma(i) untuk nilai i pada interval a dan b: a + (a+1) + (a+1+1) + ... + b, atau 0 jika interval "kosong" }

REALISASI

```
SigI(a,b) :
            if a>b then {Basis-0}
                 0
            else {Rekurens}
                 a + SigI(a+1,b)
```

DEFINISI DAN SPESIFIKASI

$$SigI3 = \sum_{i=a}^{b} i^3$$

SigI3: 2 integer → integer

{SigI3(a,b) adalah fungsi untuk menghitung Sigma(i^3) untuk nilai i pada interval a dan b: $a^3 + (a+1)^3 + (a+1+1)^3 + ... + b^3$, atau 0 jika interval "kosong" }

REALISASI

```
SigI3(a,b) :
            if a>b then {Basis-0}
                 0
            else {Rekurens}
                 a^3 + SigI3(a+1,b)
```

DEFINISI DAN SPESIFIKASI

$$SP8 = \sum_{i=a}^{b} (1/i*(i+2))$$

SP8 : 2 integer → real

{SP8(a,b) adalah fungsi untuk menghitung deret konvergen ke π/8 pada interval a dan b atau 0 jika interval "kosong". Rumus :
*1/(1*3) + 1/(5*7) + 1/(9*11) + ... }*

REALISASI

```
SP8(a,b) :
   if a>b then {Basis-0}
       0
   else {Rekurens}
        (1/((a)*(a+2)) + SP8(a+4,b)
```

Definisikan fungsi-fungsi berikut:

DEFINISI DAN SPESIFIKASI

Id : integer → integer

{ Id(i) mengirimkan nilai i)
Id(i) : i

P1 : integer → integer

{ P1(i) mengirimkan nilai i+1)
`P1(i) : i+1`

P4 : integer → integer

{ P4(i) mengirimkan nilai i+4)
`P4(i) : i+4`

Cube : integer → integer

{ Cube(i) mengirimkan nilai i^3)
`Id(i) : i^3`

T : integer → real

{ T(i) mengirimkan nilai 1/((i+1)(i+3))*
`T(i) : 1/((i+1)*(i+3))`

Definisikan suatu type numerik, yang merupakan union (gabungan) dari type integer dan type real: type numerik : union dari integer dan real

Definisikan "Sigma" dari deret :

$$\sum_{n=a}^{b} f(n) = f(a) + \ldots + f(b)$$

yang merupakan rumus umum dari penjumlahan suku deret dengan **fungsi sebagai parameter fungsi**

Definisikan fungsi "Sigma" yang umum yang dapat mewakili SigI1, SigI3 dan SP8 sebagai berikut:

DEFINISI

type numerik : union dari type integer dan real

Sigma : integer, integer,(integer → numerik),(integer → numerik) → numerik

{Sigma (a,b,f,s) adalah penjumlahan dari deret/serie f(i), dengan
mengambil nilai subseri a, s(a), s(s(a)),....
pada interval [a..b] atau 0 jika interval kosong }

REALISASI

```
Sigma(a,b,f,s) :
          if a>b then {Basis-0}
                0
           else {Rekurens}
                f(a) + Sigma(s(a),b,f,s)
```

Maka :

Sigma(a,b,Id,P1) identik dengan SigI(a,b)
Sigma(a,b,Cube,P1) identik dengan SigI3(a,b)
Sigma(a,b,T,P4) identik dengan SP8(a,b)

Id,Cube, T, P1, P4 adalah fungsi-fungsi yang akan dipakai sebagai parameter dari fungsi Sigma pada saat aplikasi, dan semuanya merupakan fungsi. Bagaimana cara memakai fungsi sebagai parameter pada saat aplikasi (run time)?

Caranya adalah dengan ekspresi LAMBDA

Perhatikan ekspresi sbb :

let a=3; b= 5+x in
max2(a,b)

dengan max2(a,b) adalah fungsi yang mengirimkan nilai maksimum dari a,b.

Ekspresi tsb. dapat ditulis : max(a,5+x)

Notasi LAMBDA memungkinkan kita memakai fungsi tanpa memberi nama seperti pada contoh di atas. Konstanta hasil fungsi dapat digunakan sebagai parameter efektif pada ekspresi fungsional

Cara penulisan ekspresi lambda untuk Id, Cube, P1, P4, T :

Id : λ x.x
Cube : λ x.x^3
P1 : λ x.x+1
P4 : λ x.x+4

Aplikasi terhadap ekspresi lambda :

Tuliskan

$\lambda x.x^3$ (2) sebagai ganti dari Cube(2)

Evaluasinya akan sama.

Maka fungsi Sigma dapat dituliskan sbb :

Untuk SigI : Sigma(a,b, λ x.x, λ x.x+1)
Untuk SigI3 : Sigma(a,b, $\lambda x.x^3$, λx.x+1)
Untuk SP8 : Sigma(a,b, λx.1/((x+1)+(x+3)), x.x+4)
Hasil evaluasi sesuai dengan type hasil ekspresi

Ekspresi lambda dengan parameter banyak dituliskan sebagai:

λx,y.x+y

adalah fungsi untuk menjumlahkan nilai x dan y.
Kedua ekspresi berikut adalah ekivalen :

λx,y. x+y
λ x. λy. x+y
(λx,y. x+ y) (2,3)
(λx. λ y. x+ y) (2,3)
(λy. 2 + y) (3)
2 + 3 = 5

Perhatikan bahwa λx,y.x+y dapat diaplikasi dengan satu parameter saja.

(λ x,y. x+y) (2)
λy. 2+y
(λ x. λy. x+y) (2) menambahkan dua ke nilai y

Fungsi Sebagai Hasil Dari Evaluasi (Range)

Perhatikan bahwa derivasi dari $f(x) = x^3$ adalah sebuah fungsi $f'(x) = 3\ x^2$, sedangkan nilai derivasi f'(x) untuk x=2 adalah suatu nilai numerik.
Maka derivasi suatu fungsi pada suatu titik dapat didefinisikan sbb:
Derivasi (real → real), real → real → real
DerivTitikX (real → real), real, real → real

Derivasi (f,dx): (f(x+dx) - f(x))/dx
derivasi untuk $f(y)= y^3$ dengan notasi lambda adalah
λ x. ((λy. y^3) (x+dx) - (λy. y^3) (x) / dx
Sehingga DerivTitikX untuk $f(y) = y^3$ dan nilai dx=0.005 dan NilaiX = 5 dituliskan sebagai aplikasi dari DerivTitikX dengan parameter

(λy.y^3, 0.005) (5)

adalah nilai f'(x) pada titik x=5

DEFINISI

Derivasi : (real → real), real → (real → real)
{Derivasi (f,dx) adalah derivasi fungsi f(x) dengan interval dx: (f(x+dx)-f(x))/dx}

DerivTitikX : ((real → real) , real) , real → real
{DerivTitikX (f,dx, NilaiX) adalah Nilai derivasi fungsi f(x) pada titik X : Aplikasi dari fungsi dengan parameter Derivasi (f,dx) dan Nilai). Karena DerivTitikX adalah aplikasi terhadap fungsi, maka Derivasi(f,dx) dapat dituliskan:
a. Realisasi-1: dengan hanya melakukan aplikasi terhadap Derivasi
b. Realisasi-2 : menuliskan realisasinya dengan ekspresi lambda dan akan diinstansiasi dengan NilaiX seperti pada realisasi kedua}

REALISASI-1

```
Derivasi(f,dx) :
          ( f(x+dx) - f(x) )/ dx
```

APLIKASI

```
DerivTitikX (Derivasi(f,dx),NilaiX)
```

REALISASI-2 (DENGAN EKSPRESI LAMBDA)

```
Derivasi(f,dx) :
          ( f(x+dx) - f(x) )/ dx
DerivTitikX (f,dx, NilaiX) :
          λ.x ( f(x+dx) - f(x) )/ dx (NilaiX)
```

Catatan :

- Terjemahan fungsi diatas tidak mudah, dan sangat spesifik. Lihat bku bagian kedua

Ekspresi Lambda Dengan Lebih Dari Satu Parameter

Ekspresi lambda dengan parameter banyak dituliskan sebagai:

λx,y.x+y

adalah fungsi untuk menjumlahkan nilai x dan y.

Kedua ekspresi berikut adalah ekivalen :

λx,y. x+y
λ x. λy. x+y
(λx,y. x+ y) (2,3)
(λx. λ y. x+ y) (2,3)
(λy. 2 + y) (3)
2 + 3 = 5

Perhatikan bahwa λx,y.x+y dapat diaplikasi dengan satu parameter saja.

(λ x,y. x+y) (2)
λy. 2+y
(λ x. λy. x+y) (2) menambahkan dua ke nilai y

Static and dynamic binding

Perhatikan ekspresi sebagai berikut :

let n=3 in
let f = λx.x+n in
let n=2 in f(4)

Ekspresi lambda di atas berarti : tambahkan n pada f

Dengan static binding (pada saat definisi) :

n = 3 dan hasilnya adalah tambahkan 3 pada 4 berarti 7

Dengan dynamic binding (pada saat aplikasi) :

n = 2 dan hasilnya adalah tambahkan 2 pada 4 berarti 6

Anda harus memperhatikan binding macam apa yang dilakukan oleh interpreter, supaya hasil fungsi seperti yang diharapkan.

Studi kasus lewat contoh

Contoh-1 : OFFset (Ekspresi lambda dengan hasil numerik)

Persoalan :

Tuliskanlah definisi, spesifikasi dan realisasi sebuah fungsi yang melakukan "offset" atau penggeseran terhadap elemen list, dan menghasilkan sebuah list baru hanya berupa elemen yang digeser sesuai dengan delta yang diberikan ketika melakukan offset.
Contoh :

- Diberikan sebuah list integer, dengan fungsi Offset Plus 2, maka hasilnya adalah sebuah list baru yang elemennya berupa integer tapi setiap elemen sudah bertambah dengan dua.
- Diberikan sebuah list integer, dengan fungsi Offset Minus 1, maka hasilnya adalah sebuah list baru yang elemennya berupa integer tapi nilai setiap elemen sudah berkurang dengan satu.
- Diberikan sebuah list of integer, dengan fungsi offset yang tergantung kepada nilai elemen yang akan dioffset, maka hasilnya adalah sebuah list integer yang setiap elemennya diubah

Nilai Elmt	Offset
0-40	10
41-60	5
61- 80	3
>80	1
lainnya	0

OFFSETLIST **OffSetList(List,OffSet)**

DEFINISI DAN SPESIFIKASI

OffSetList : list of integer tidak kosong, OffSet → list of integer

{OffSetList (Li,OffSet), dengan Li adalah list integer dan OffSet adalah sebuah fungsi dengan definisi melakukan offset. OffSetList menghasilkan sebuah list integer dengan elemen yang sudah dioffset}

REALISASIP

```
OffSetList(Li, OffSet) :
    Konso (OffSet(FirstElmt(Li), OffSetList(Tail(Li),OffSet) )
```

BEBERAPA CONTOH OFFSET

{ f adalah Plus 2 }

```
OffSet  ≡ Plus2(i) : i + 2
```

{ f adalah Minus 1 }

```
OffSet ≡ Minus1(i) : i - 1
```

{f adalah ekspresi kondisional }

```
   OffSet ≡  OffKond(i): depend on i
                           0 ≤  i ≤   40 : 10
                           41 ≤  i ≤   60 : 5
                           61 ≤  i ≤   89 : 3
                                 i >   89 : 1
                            else          : 0
```

APLIKASI

```
⇒ OffSetList([1,3,6,0,-9,45], λ.i, i+2  )
⇒ OffSetList([1,3,6,0,-9,45], λ.i, i-1  )
⇒ OffSetList([31,1,3,26,0], λ.i, i+2  )
```

Contoh-2 : Filter (Ekspresi lambda dengan hasil boolean)

Persoalan :

Tuliskanlah definisi, spesifikasi dan realisasi sebuah fungsi yang melakukan "filter" atau penyaringan terhadap elemen list, dan menghasilkan sebuah list baru hanya berupa elemen yang lolos dari kriteria yang ada pada filter, yaitu sebuah fungsi yang ekspresinya adalah ekspresi boolean. Contoh :

Diberikan sebuah list integer, dengan filter fungsi positif, maka hasilnya adalah sebuah list baru yang elemennya hanya berupa integer positif.

Diberikan sebuah list integer, dengan filter fungsi negatif, maka hasilnya adalah sebuah list baru yang elemennya hanya berupa integer negatif.

FILTERLIST **FilterList(List,f)**

DEFINISI DAN SPESIFIKASI

FilterList : list of integer tidak kosong , f → list of integer

{FilterList (Li, f), dengan Li adalah list integer dan f adalah sebuah predikat dengan definisi f(i) menghasilkan sebuah list integer dengan elemen yang memenuhi Predikat f}

REALISASI

```
   filterList(Li, f) :
       if not f(FirstElmt(Li)
       then filterList(Li, f)
       else Konso(FirstElmt(Li), filterList(Li, f) )
```

BEBERAPA CONTOH FUNGSI F

{ filter adalah integer positif : IsPos? (i) benar jika i positif }

```
   f ≡ IsPos(i) : i>0 )
```

{ filter adalah integer positif : IsNeg? (i) benar jika i negatif }

```
   f ≡ IsNeg(i): i<0 )
```

{ filter adalah: Kabisat?(i) : bilangan kelipatan 4 tapi bukan kelipatan 100 }

```
   f ≡ Kabisat(i) ;. (i mod 4 = 0) and (i mod 100 ≠ 0) )
```

APLIKASI

```
⇒ FilterList([1,3,6,0,-9,45], IsPos)
```

```
⇒ FilterList([-1,3,-6,0,-9,45], IsNeg)
⇒ FilterList([31,1,3,26,0], IsKabisat ))
```

APLIKASI

```
⇒ FilterList([1,3,6,0,-9,45], λ.i, i>0)
⇒ FilterList([-1,3,-6,0,-9,45], λ.i, i<0)
⇒ FilterList([31,1,3,26,0], λ.i, (i mod 4 = 0) and (i mod 100 ≠ 0))
```

Contoh-3 : FilterRekList : Ekspresi lambda rekursif

Persoalan :

Tuliskanlah sebuah fungsi yang melakukan linearisasi sebuah list of list integer menjadi list integer, dan atom yang dijadikan anggota dari list integer hasil adalah atom yang menjadi anggota dari suatu list yang dihasilkan oleh suatu fungsi f

.

FilterRekLIST	**FilterRekList(List,f)**

DEFINISI DAN SPESIFIKASI

FilterRekList : list of list integer tidak kosong , fungsi → list of integer

{ FilterRekList (Si, f), dengan Si adalah list of list integer dan f adalah sebuah fungsi dengan definisi diberikan suatu list of list integer menghasilkan list integer yang menjadi anggota list yang dihasilkan oleh f}

REALISASI

```
FilterRekList(Si, f) :
  Konso (f(FirstList(Si), FilterRekList(TailList(Si),f))
```

BEBERAPA CONTOH F

{Contoh ekspresi f(S) adalah sebuah fungsi yang menerima sebuah list of list integer dan menghasilkan list integer berasal dari atom-atom list }

```
   f(S) ≡ LINEAR (S)
            if Isatom? (FirstList(S))
            then Konso(FirstList(S), LINEAR(TailList(S))
            else { bukan atom, harus dilinearkan }
                Konso(LINEAR(FirstList(S),LINEAR(TailList(S))
```

{Contoh ekspresi FilterRek(S) adalah sebuah fungsi yang menerima sebuah list of list integer dan menghasilkan list integer berasal dari atom-atom list, hanya jika atomnya positif }

```
   f(S) ≡
            if Isatom? (FirstList(S)
            then if FirstList(S) > 0  then FirstList(S)
                 else []
            else { bukan atom, harus dilinearkan }
                  Konso(FilterRek(FirstList(S), FilterRek(TailList(S))
```

APLIKASI

⇒

1. Perhatikanlah bahwa karena dalam pemakaian ekspresi lambda kita menuliskan ekspresi secara langsung, maka ekspresi rekursif sebagai ekspresi **lambda tidak mungkin** dilakukan karena ekspresi rekursif membutuhkan nama untuk aplikasi.
2. Jadi **ekspresi lambda tidak boleh rekursif**.