Analisis Rekurens dan Struktur Data Rekursif

Tim Pengajar

IF1210 Dasar Pemrograman

ANALISIS REKURENS

The Handshake Problem

Ada *n* orang di dalam sebuah ruangan. Jika masing-masing orang harus bersalaman dengan setiap orang lainnya, ada berapa *handshakes* **h(n)** yang terjadi?

Bagaimana cara memecahkan persoalan ini?

Latihan: The Handshake Problem Ide penyelesaian

- Setiap orang dapat berpartisipasi untuk menyelesaikan persoalan ini.
  - Bagaimana “versi kecil” dari persoalan yang dapat dibantu penyelesaiannya?

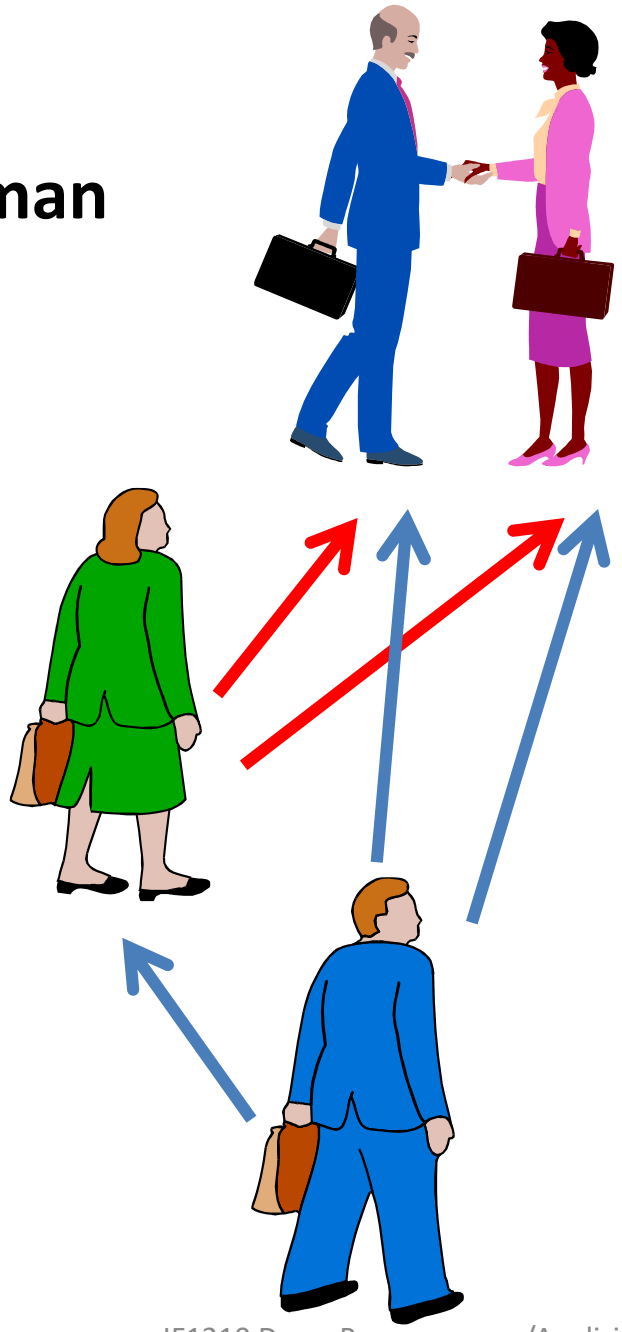
1 kali salaman

+

2 kali salaman

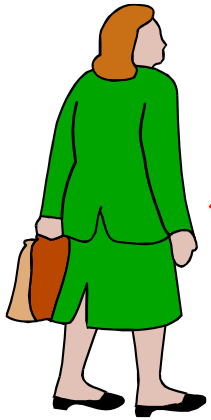
+

3 kali salaman



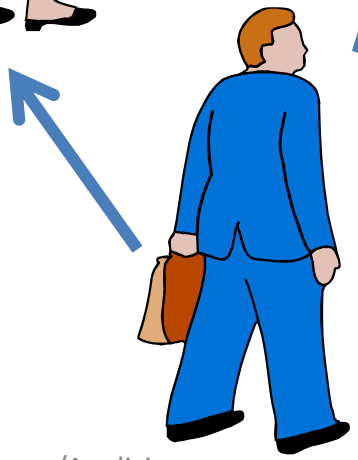
H(2) = 1

H(3) = H(2) + 2



H(n) = ?

H(4) = H(3) + 3



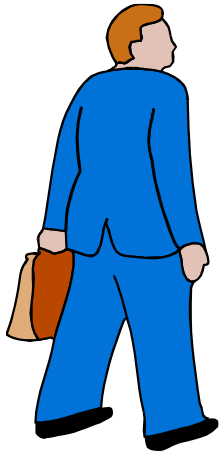
H(n) = H(n-1)+n-1

h(n) = h(n-1) + n-1

h(4) = h(3) + 3

h(3) = h(2) + 2

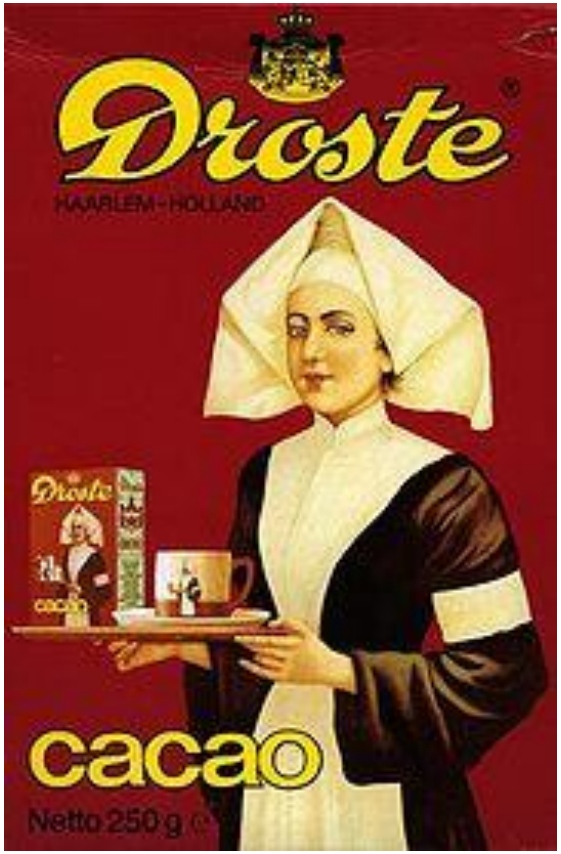
Kasus basis:
h(2) = 1



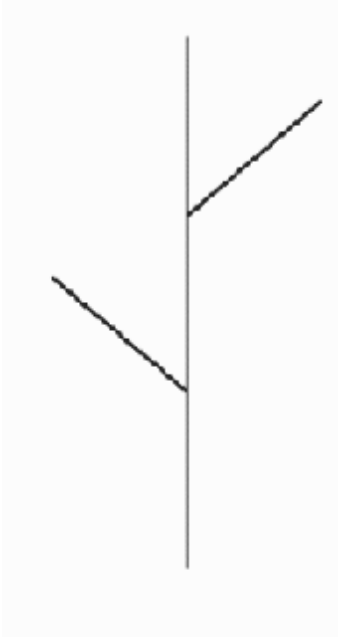
h(n) => Sum dari integer mulai 1 hingga n-1
= n(n-1) / 2

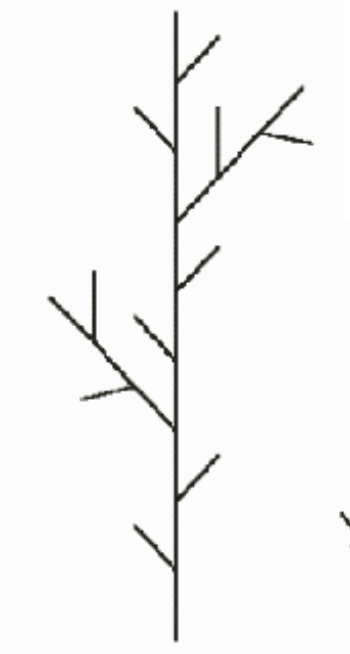
Recursion is all about breaking a big problem into smaller occurrences of that same problem.

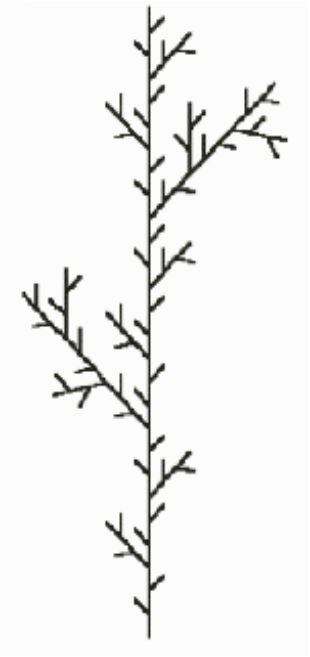
Rekursifitas

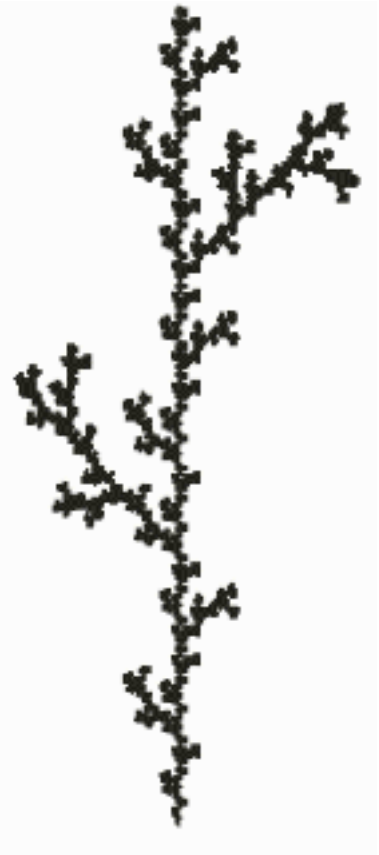


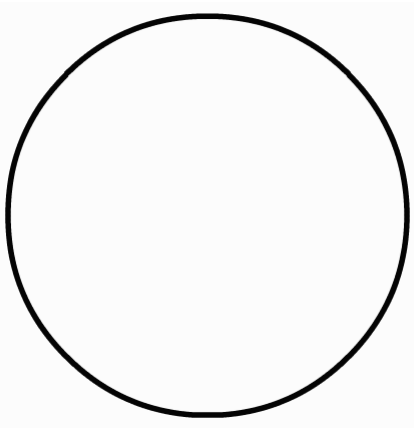
- Definisi:
  - Suatu entitas disebut **rekursif** jika pada definisinya terkandung terminologi dirinya sendiri

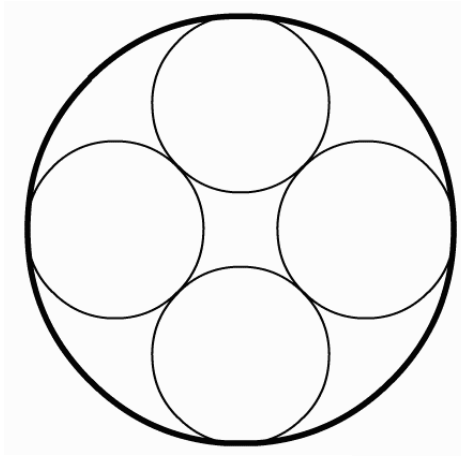


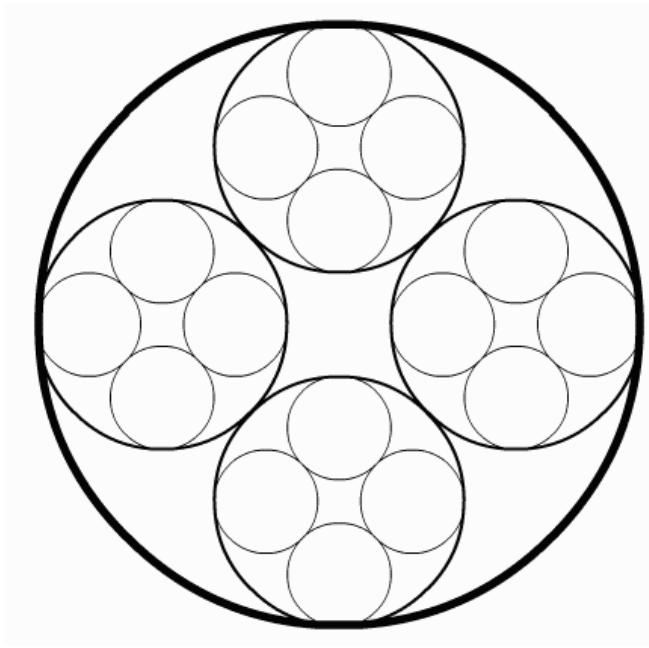


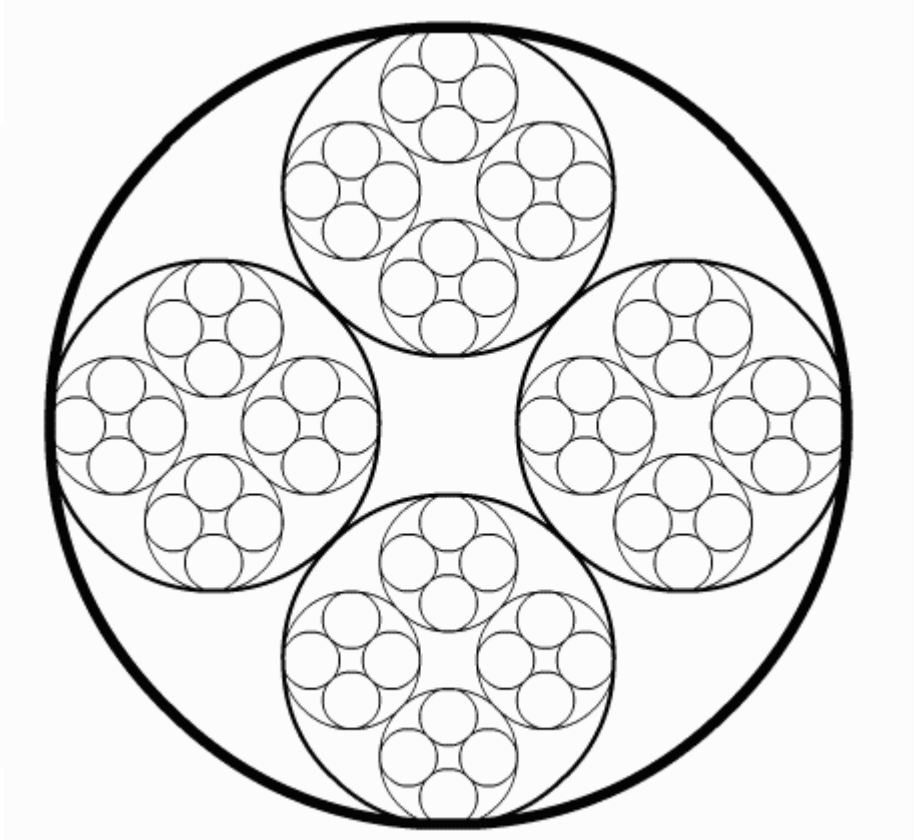


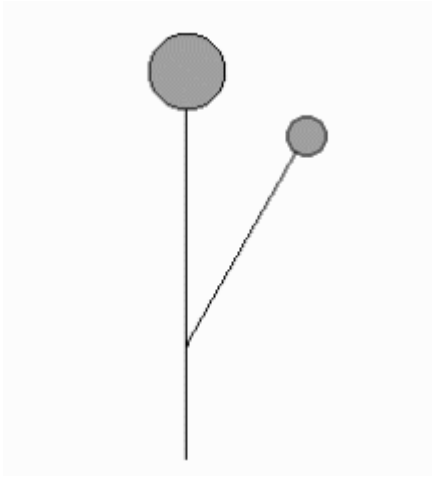




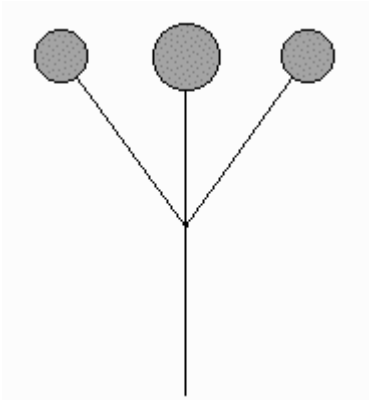




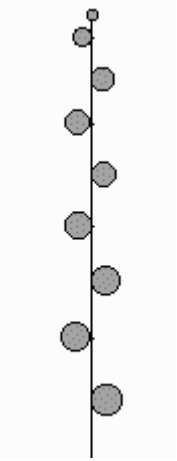




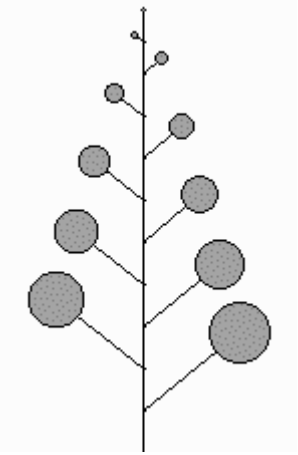
monochasium



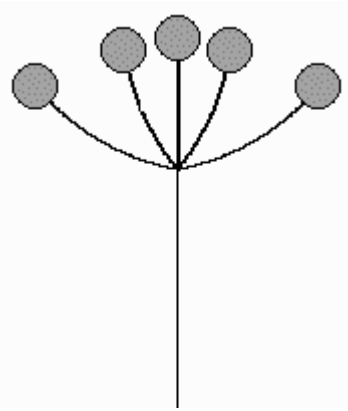
dichasium



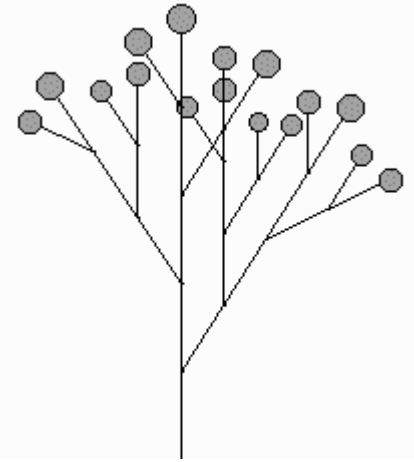
spike



raceme



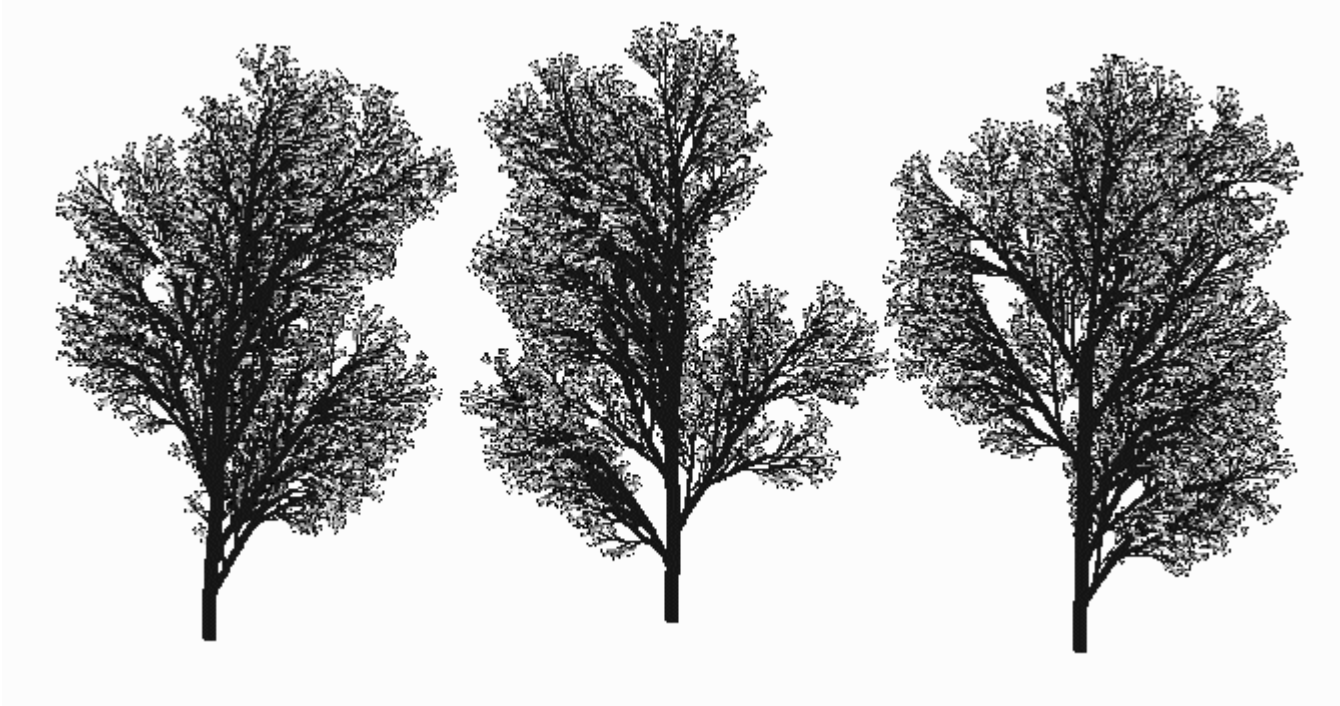
umbel



panicle

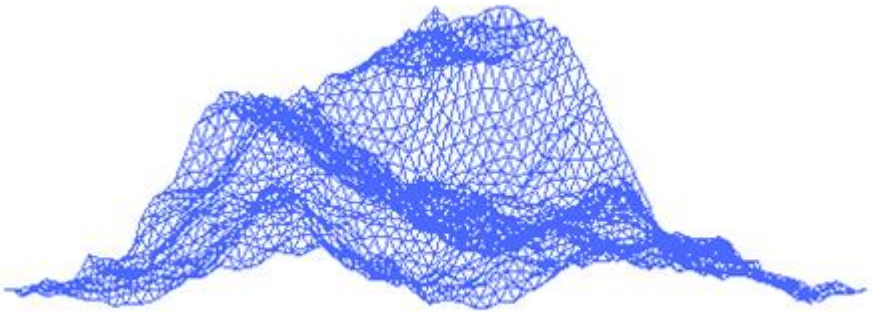
Plant Forms

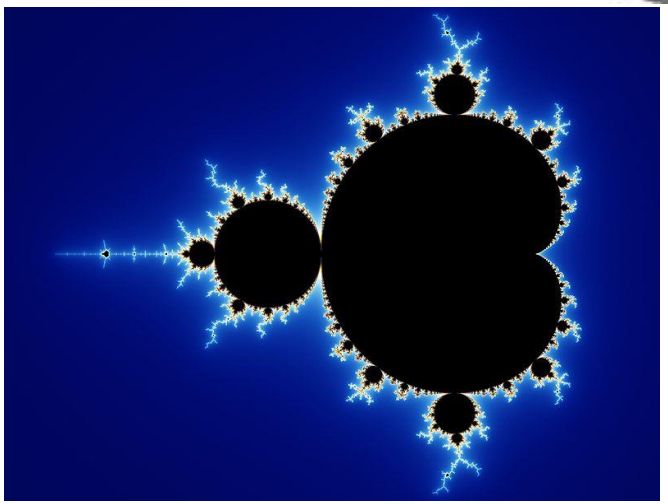
some common inflorescences



Trees modeled using deReffye's method

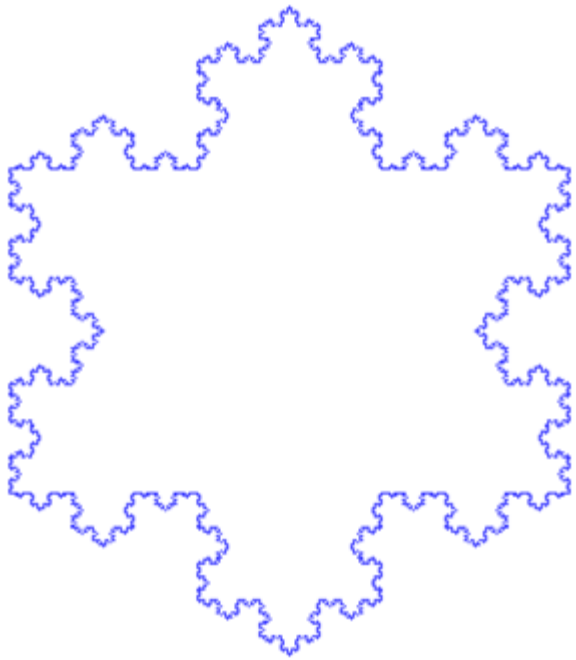
Contoh-contoh gambar fraktal





Aplikasi Rekursifitas dalam bidang informatika

- **Fraktal**:
  - bentuk-bentuk geometris yang terdiri atas bagian-bagian kecil, tiap bagian adalah kopi (dalam ukuran yang lebih kecil) dari bentuk keseluruhan
  - Biasanya diimplementasikan dari fungsi matematis yang bersifat rekursif



Ekspresi Rekursif

- Definisi entitas (type, fungsi) disebut rekursif jika definisi tersebut mengandung terminologi dirinya sendiri. (diktat hlm. 46)
- Ekspresi rekursif direalisasikan dengan membuat fungsi rekursif dan didasari analisis rekurens.

Akan dibahas pada pertemuan berikutnya

Analisis Rekurens

- Solusi rekursif terdiri dari dua bagian:
  - **Basis**, kasus yang menyebabkan fungsi berhenti, karena jawaban sudah bisa diperoleh
  - Bagian **rekurens** : mengandung call terhadap fungsi tersebut (aplikasi dari fungsi), dengan parameter bernilai mengecil (menuju basis).
- Solusi rekursif memiliki sekurang-kurangnya satu kasus basis dan satu kasus rekursif.
  - Dimungkinkan ada lebih dari satu kasus basis maupun rekursif.
- Bagian terpenting dari analisis rekurens adalah mengidentifikasi kasus-kasus ini.

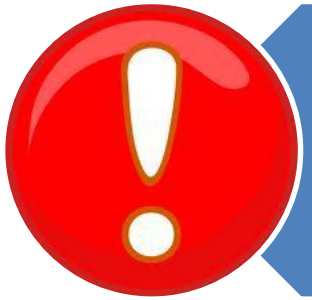
Contoh Lain…

- Bagaimana caranya kita memisahkan menjadi 2 bagian yang sama permen M&M dari mangkok besar ini, tanpa harus **terlebih dahulu** menumpahkan seluruh isinya atau menghitung jumlahnya?
  - Bagaimana jika beberapa orang membantu memecahkan persoalan ini? Bisakah tiap orang melakukan sebagian kecil dari pekerjaan?
    - Berapa jumlah M&M yang mudah dibagi menjadi dua (bahkan kalau untuk anak kecil yang tidak bisa menghitung?)

Kerangka Fungsi Rekursif (Notasi Haskell dengan Guard)

```
<fungsi> <list-parameter>
   | <kondisi-basis>    = <ekspresi-1 >
   | <kondisi-rekurens>= <fungsi>(<ekspresi-2>)

Catatan:
<kondisi-rekurens> bisa menggunakan
"otherwise"
```

Nilai parameter mengecil menuju basis

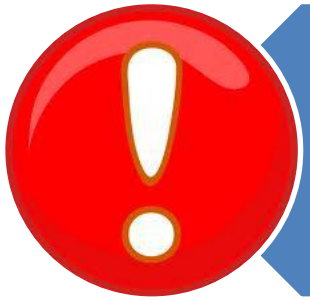


Tulislah secara eksplisit dalam teks program anda: mana bagian basis, mana rekurens berupa komentar dalam program

Kerangka Fungsi Rekursif (Notasi Haskell dengan if-then-else)

```
<fungsi> <list-parameter> =
  if <kondisi-basis>  then <ekspresi-1 >
  else <fungsi>(<ekspresi-2>) --kondisirekurens
```

Nilai parameter
mengecil
menuju basis



Tulislah secara eksplisit dalam teks program anda: mana bagian basis, mana rekurens berupa komentar dalam program

Studi Kasus 1: Definisi Faktorial

- Definisi 1:

  0! : 1

  N! : N * (N-1) !

- Definisi 2:

  1! : 1

  N! : N * (N-1) !

Nilai N mengecil menuju basis

- Definisi 3:

  1! : 1

  N! : (N+1)! / (N+1)

Nilai N tidak pernah mencapai basis → TIDAK bisa diimplementasi menjadi fungsi rekursif


```
-- Faktorial (definisi-1)                                    (n)
```

Jika domain nilai terkecil 0,
Menggunakan basis n=0

```
-- DEFINISI DAN SPESIFIKASI
fac :: Int -> Int
  {- fac(n) = n! sesuai dengan definisi rekursif factorial,
     dengan basis n = 0 -}
```

```
-- REALISASI { menggunakan Guard }
fac n
    | n == 0    = 1                 -- Basis
    | otherwise = fac (n-1) * n     -- Rekurens
```

Terdapat aplikasi fungsi fac di dalam realisasi → rekurens

```
-- Faktorial2 (definisi-2)                                    fac2(n)
```

```
-- DEFINISI DAN SPESIFIKASI
fac2 :: Int -> Int
  {- fac2(n) = n! sesuai dengan definisi rekursif factorial,
     dengan basis 1 -}
```

```
-- REALISASI menggunakan if-then-else
fac2 n = if n==1 then          -- Basis
             1
         else                  -- Rekurens
             fac2 (n-1) * n
```

Karena nilai terkecil dari domain adalah 1, maka harus menggunakan basis n = 1

Studi Kasus 2: **FPB(m,n)**
Faktor Persekutuan Terbesar

- **m,n: integer>0**
- Alternatif solusi 1: hasil kali faktor prima yang sama dari m dan n → pendekatan prosedural
- Alternatif solusi 2: fpb ≤ min(m,n)
  - Cari minimum m,n → kurangi satu sampai ketemu bilangan yang merupakan pembagi m dan n
- Alternatif solusi 3:
  fpb(m,n) = fpb(n,m mod n)

```
-- FPB (solusi-3)                                    fpb(m,n)
```

```
-- DEFINISI DAN SPESIFIKASI
fpb :: Int -> Int -> Int
  {- fpb(m,n) menghasilkan bilangan integer terbesar yang
     dapat membagi m dan n tanpa sisa -}
```

```
-- REALISASI

  fpb m n =
     if ((mod m n)==0) then    -- Basis
        n
     else                      -- Rekurens
        fpb n (mod m n)
```

Ide: fpb(m,n) = fpb(n,m mod n).
Kasus basis?

Jika n habis membagi m, maka (m mod n) = 0 dan fpb = n

Latihan Soal

Buatlah definisi, spesifikasi, dan realisasi dari fungsi-fungsi berikut (menggunakan pendekatan rekursif):

1. **DeretSegitiga**, merupakan fungsi untuk mencari nilai bilangan ke-n pada deret segitiga.
   Deret segitiga: 1, 3, 6, 10, 15, …
2. **IsGanjil**, merupakan predikat untuk memeriksa apakah sebuah bilangan integer (≥ 0) merupakan bilangan ganjil.
3. **LuasBS**, merupakan fungsi untuk menghitung luas bujur sangkar dengan panjang sisi tertentu.

Latihan Soal

4. SumOfDigits, menghitung hasil penjumlahan dari setiap bilangan tunggal yang terdapat di dalam sebuah bilangan integer positif.

   Misalnya:

   - SumOfDigits(234) = 2 + 3 + 4 = 9
   - SumOfDigits(38) = 3 + 8 = 11
   - SumOfDigits(5) = 5

   Apabila didefinisikan bahwa SumOfDigits dari bilangan negatif dilakukan dengan mengabaikan tanda '-', buatlah fungsi SumOfDigitsPosNeg yang menangani hal ini.

   Misalnya: SumOfDigitsPosNeg(-45) = 4 + 5 = 9

Latihan Soal

5. Buatlah fungsi **sumRange** yaitu fungsi yang menerima masukan 2 bilangan integer, misalnya a dan b yang menyatakan rentang bilangan dengan syarat: $a \leq b$; a dan b bilangan positif; dan menghasilkan penjumlahan semua bilangan dari a s.d. b.

   Harus dikerjakan secara rekursif.

   Contoh aplikasi:

   – SumRange(2,2) = 2
   – SumRange(2,4) = 2+3+4 = 9

Kenapa Membuat Program Rekursif?

- Persoalan memang mengandung definisi “rekursif”.
  Contoh: **faktorial**.
- Mengolah data bertype rekursif.
  Contoh: **pemrosesan list** → **materi berikutnya**
- Programmernya ingin menulis solusi rekursif. Persoalan yang tidak rekursif tidak perlu diselesaikan secara rekursif, tapi bisa diselesaikan secara rekursif. Ada banyak jalan ke Roma……
  Contoh: **fpb (Faktor Persekutuan Terbesar)**

Mencari Solusi Rekursif

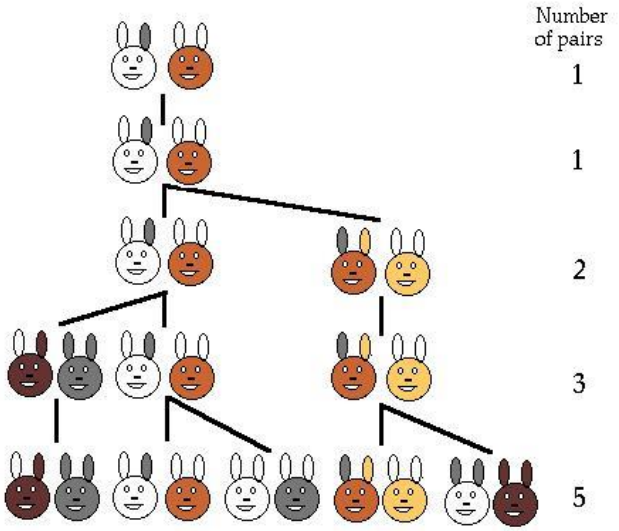
- Solusi rekursif karena definisi persoalan rekursif
  - Baca baik-baik persoalan, dan buat definisi rekursif sesuai definisi persoalan
- Solusi rekursif karena persoalan diwakili oleh struktur data rekursif → **materi berikutnya**
- Solusi rekursif untuk persoalan non rekursif
  - Buat fungsi rekursif dengan parameter

Intermezzo: How many pairs of rabbits can be produced from a single pair in a year's time?

- Asumsi:
  - Setiap pasang kelinci akan memberikan sepasang anak setiap bulan;
  - setiap pasang kelinci baru akan mulai melahirkan setelah berusia dua bulan;
  - tidak ada kelinci yang mati pada tahun itu.

- Contoh:
  - Setelah 1 bulan akan ada 2 pasang kelinci;
  - setelah 2 bulan akan ada 3 pasang kelinci;
  - setelah 3 bulan akan ada 5 pasang kelinci (karena pasangan yang lahir pada bulan pertama sudah akan mulai melahirkan anak).

Population Growth in Nature



- Leonardo Pisano (Leonardo Fibonacci = Leonardo, son of Bonaccio) proposed the sequence in 1202 in *The Book of the Abacus*.
- Fibonacci numbers are believed to model nature to a certain extent, such as Kepler's observation of leaves and flowers in 1611.

Fibonacci Numbers

- Fibonacci numbers:
  `0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...`
  where each number is the sum of the preceding two.

- Recursive definition:
  - `F(0) = 0;`
  - `F(1) = 1;`
  - `F(number) = F(number-1)+ F(number-2);`

LIST SEBAGAI STRUKTUR DATA REKURSIF

Type

- Type adalah himpunan nilai dan sekumpulan operator yang terdefinisi terhadap type tersebut
  - Dalam konteks fungsional: operator dijabarkan dalam bentuk fungsi
- Jenis-jenis type:
  - Type dasar → sudah tersedia: integer, real, character, boolean
  - Type bentukan → dibuat sendiri
- Jenis type dari segi banyaknya elemen yang harus dikelola:
  - Type yang mengelola satu objek, contoh: integer, real, point, jam, dll.
  - Type koleksi/kumpulan objek, contoh: array of integer, list of character, dll.

Mendefinisikan Type

- Dalam konteks fungsional, mendefinisikan type adalah mendefinisikan:
  - Nama dan struktur type (komponen-komponennya)
  - Selektor untuk mengakses komponen-komponen type
  - Konstruktor untuk “membentuk” type
  - Predikat untuk menentukan karakteristik dan pemeriksaan besaran
  - Fungsi-fungsi lain yang didefinisikan untuk type tersebut

Tipe Rekursif

- Tipe rekursif :
  - Jika teks yang mendefinisikan tipe mengandung referensi terhadap diri sendiri, maka disebut tipe rekursif.
  - Tipe dibentuk dengan komponen yang merupakan tipe itu sendiri.

Contoh Tipe Rekursif:
Bilangan Integer

bilangan integer

Basis : 0 adalah bilangan integer

Rekurens : if x adalah bilangan integer

then x+1 adalah bilangan integer

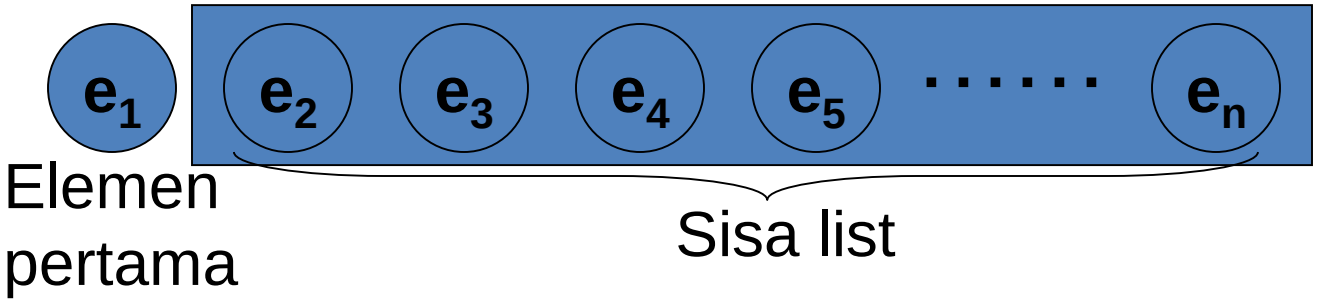
bilangan integer ganjil

Basis : 1 adalah bilangan integer ganjil

Rekurens : if x adalah bilangan integer ganjil

then x+2 adalah bilangan integer ganjil

Definisi List



- List adalah sekumpulan elemen yg bertipe sama; disebut juga sequence atau series
- Tipe rekursif
  - Basis 0: list kosong adalah sebuah list
  - Rekurens: list terdiri dari sebuah elemen dan sublist (sublist juga bertipe list)

LIST dlm Kehidupan Sehari-hari

- Dalam kehidupan sehari-hari, list merepresentasi:
  - Teks (list of kata)
  - Kata (list of huruf)
  - Sequential file (list of record)
  - Table (list of elemen tabel, cth utk tabel integer: list of integer)
  - List of atom simbolik (dalam LISP)

LIST dlm Dunia Pemrograman

- Dalam dunia pemrograman
  - Antarmuka basis grafis (GUI): list of windows, list of menu items, list of button, list of icon
  - Program editor gambar: list of figures
  - Program pengelola sarana presentasi: list of slides
  - Program pengelola spreadsheet: list of worksheet, list of cell
  - Sistem operasi: list of terminal, list of job

JENIS LIST

- LIST dg elemen sederhana
  - LIST dg elemen bilangan integer
  - LIST dg elemen karakter (teks)
  - LIST dg elemen type bentukan, cth: list of point (tidak dibahas di kuliah ini)
- LIST dg elemen list (disebut list of list)
  - (tidak dibahas di kuliah ini)

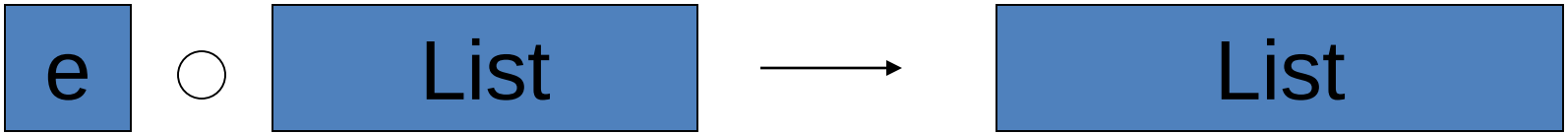
List dengan Elemen Sederhana

Definisi rekursif:

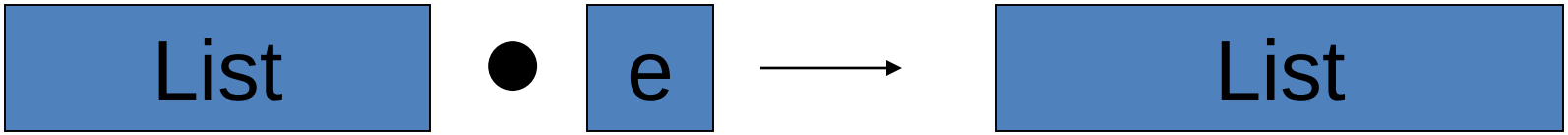
- **Basis**: list kosong adalah sebuah list
- **Rekurens**: list dapat dibentuk dengan menambahkan elemen pada list (**konstruktor**), atau terdiri dari sebuah elemen dan sisanya adalah list (**selektor**)
  - Elemen list: dapat berupa type dasar (integer, character, dll) dan type bentukan (Point, Jam, dll)

DEFINISI & SPESIFIKASI LIST

- **type** List: [ ] atau [e o List]

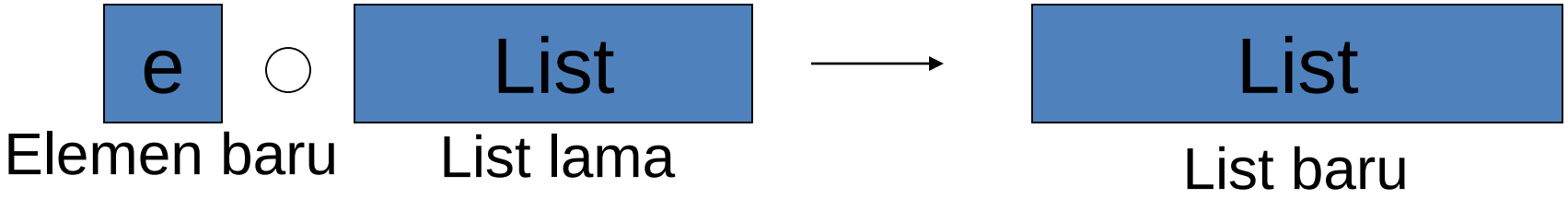


- **type** List: [ ] atau [List • e]



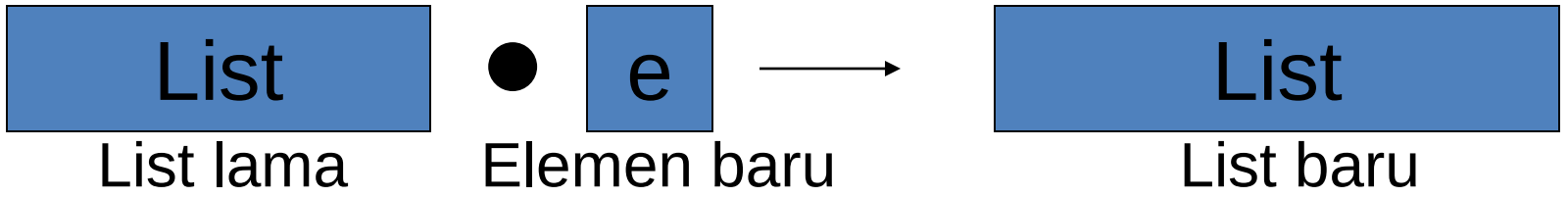
KONSTRUKTOR

- **Konso** : elemen, List → List



Konso (5,[1,3,6,7]) → [5,1,3,6,7]

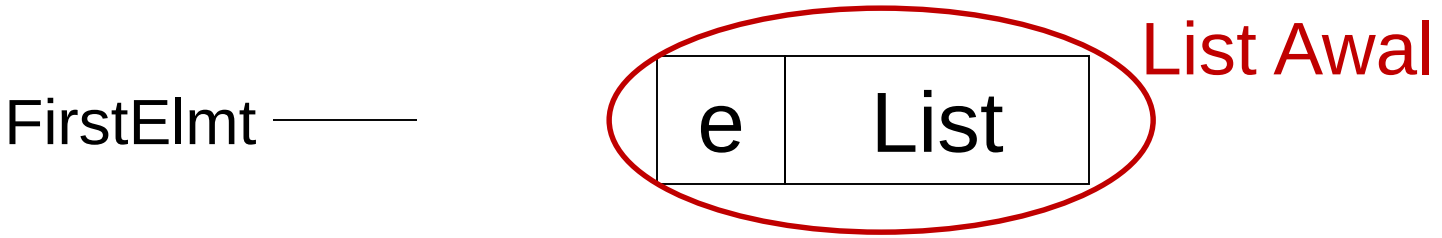
- **Kons•** : List, elemen → List



Kons• ([1,3,6,7],5) → [1,3,6,7,5]

SELEKTOR

- **FirstElmt**: List tidak kosong → elemen



FirstElmt([5,1,3,6,7]) = 5

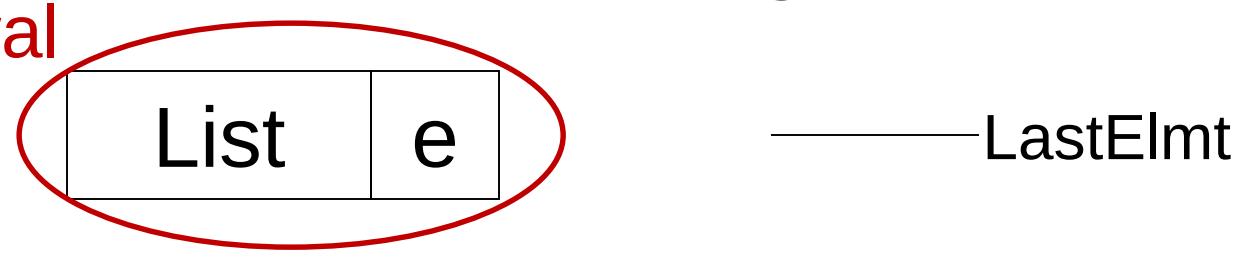
- **Tail**: List tidak kosong → List

Tail([5,1,3,6,7]) = [1,3,6,7]

SELEKTOR

- **LastElmt**: List tidak kosong → elemen

List Awal



LastElmt([5,1,3,6,7]) = 7

- **Head**: List tidak kosong → list

Head([5,1,3,6,7]) = [5,1,3,6]

PREDIKAT DASAR

- **Pemeriksaan Basis-0**

  IsEmpty: List → boolean

  { IsEmpty(L) menghasilkan true jika list L kosong }

- **Pemeriksaan Basis-1**

  IsOneElmt: List → boolean

  { IsOneElmt(L) menghasilkan true jika list L hanya berisi 1 elemen }

List Pada Haskell

- Contoh list di Haskell
  - List of integer [Int]
    [1,2,3,2,4,1]
  - List of Char [Char]
    [‘a’,‘e’,‘2’]
  - List of Point [(Int,Int)]
    [(1,2),(0,0),(-2,9)]

Deklarasi List of <type_elemen>

- Harus dituliskan sebagai komentar, ganti <type_elemen> dengan type elemen list (integer, character, dll.)

```
-- DEFINISI DAN SPESIFIKASI LIST

-- type List of <type_elemen>: [ ] atau [e o List]
-- Definisi type List of <type_elemen>
-- Basis: List of <type_elemen> kosong adalah list of <type_elemen>
-- Rekurens: List tidak kosong dibuat dengan menambahkan sebuah
-- elemen bertype <type_elemen> di awal sebuah list

-- type List of <type_elemen>: [ ] atau [List • e]
-- Definisi type List of <type_elemen>:
-- Basis: List of <type_elemen> kosong adalah list of <type_elemen>
-- Rekurens: List tidak kosong dibuat dengan menambahkan sebuah
-- elemen bertype <type_elemen> di akhir sebuah list
```

Konstruktor List

```
-- DEFINISI DAN SPESIFIKASI KONSTRUKTOR

konso :: <type_elemen> -> [<type_elemen>] -> [<type_elemen>]
-- konso(e,l) menghasilkan sebuah list dari e (sebuah
-- elemen) dan l (list of elemen),
-- dengan e sebagai elemen pertama: e o l -> l'
-- REALISASI
konso e l = [e] ++ l


konsDot :: [<type_elemen>] -> <type_elemen> -> [<type_elemen>]
-- konsDot(l,e) menghasilkan sebuah list dari l (list of
-- elemen) dan e (sebuah elemen),
-- dengan e sebagai elemen terakhir: l • e -> l'
-- REALISASI
konsDot l e = l ++ [e]
```

Konstruktor List of Integer

```
-- DEFINISI DAN SPESIFIKASI KONSTRUKTOR

konso :: Int -> [Int] -> [Int]
-- konso(e,li) menghasilkan sebuah list dari e (sebuah
-- integer) dan li (list of integer),
-- dengan e sebagai elemen pertama: e o li -> li'
-- REALISASI
konso e li = [e] ++ li


konsDot :: [Int] -> Int -> [Int]
-- konsDot(li,e) menghasilkan sebuah list dari li (list
-- of integer) dan e (sebuah integer),
-- dengan e sebagai elemen terakhir: li • e -> li'
-- REALISASI
konsDot li e = li ++ [e]
```

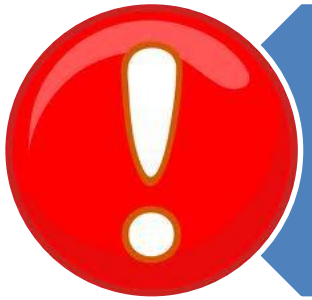
Konstruktor List of Character (Teks)


```
-- DEFINISI DAN SPESIFIKASI KONSTRUKTOR

konso :: Char -> [Char] -> [Char]
-- konso(e,lc) menghasilkan sebuah teks dari e
-- (sebuah character) dan lc (teks), dengan e sebagai
-- elemen pertama: e o L -> L'
-- REALISASI
konso e lc = [e] ++ lc


konsDot :: [Char] -> Char -> [Char]
-- konsDot(lc,e) menghasilkan sebuah teks dari lc
-- (teks) dan e (sebuah character), dengan e sebagai
-- elemen terakhir: lc • e -> lc'
-- REALISASI
konsDot lc e = lc ++ [e]
```

Selektor List

- Selektor List hanya terdefinisi pada List yang tidak kosong.

Notasi Fungsional	Notasi Haskell
FirstElmt(L)	head l
Tail(L)	tail l
LastElmt(L)	last l
Head(L)	init l



Karena *tail* dan *head* adalah fungsi yang sudah terdefinisi pada Haskell, maka untuk selanjutnya Selektor List yang digunakan sesuai Notasi Haskell

Selektor List of <type_elemen>

- Definisi dan spesifikasi selektor ditulis dalam bentuk komentar, menggunakan nama-nama fungsi manipulasi list Haskell

```
-- DEFINISI DAN SPESIFIKASI SELEKTOR

-- head : [<type_elemen>] -> <type_elemen>
-- head(l) menghasilkan elemen pertama list l, l tidak kosong

-- tail : [<type_elemen>] -> [<type_elemen>]
-- tail(l) menghasilkan list tanpa elemen pertama list l,
-- l tidak kosong

-- last : [<type_elemen>] -> <type_elemen>
-- last(l) menghasilkan elemen terakhir list l, l tidak kosong

-- init : [<type_elemen>] -> [<type_elemen>]
-- init(l) menghasilkan list tanpa elemen terakhir list l,
-- l tidak kosong
```

Contoh Pemanfaatan Selektor

```
*Main> head [1,2,3,4]      -- FirstElmt
1
*Main> tail [1,2,3,4]      -- Tail
[2,3,4]
*Main> last [1,2,3,4]      -- LastElmt
4
*Main> init [1,2,3,4]      -- Head
[1,2,3]
```

Predikat List

```
-- DEFINISI DAN SPESIFIKASI PREDIKAT

isEmpty :: [<type_elemen>] -> Bool
-- isEmpty(l) true jika list of elemen l kosong
-- REALISASI
isEmpty l = null l

isOneElmt :: [<type_elemen>] -> Bool
-- isOneElmt(l) true jika list of integer l hanya
-- mempunyai satu elemen
-- REALISASI
isOneElmt l = (length l) == 1
```

```
*Main> isEmpty [1,2,3,4]
False
*Main> isEmpty []
True
*Main> isOneElmt []
False
*Main> isOneElmt [1]
True
```

Predikat List of Integer

```
-- DEFINISI DAN SPESIFIKASI PREDIKAT

isEmpty :: [Int] -> Bool
-- isEmpty(li) true jika list of integer li kosong
-- REALISASI
isEmpty li = null li

isOneElmt :: [Int] -> Bool
-- isOneElmt(li) true jika list of integer li hanya
-- mempunyai satu elemen
-- REALISASI
isOneElmt li = (length li) == 1
```

Predikat List of Character (Teks)

```
-- DEFINISI DAN SPESIFIKASI PREDIKAT

isEmpty :: [Char] -> Bool
-- isEmpty(lc) true jika list of integer lc kosong
-- REALISASI
isEmpty lc = null lc

isOneElmt :: [Char] -> Bool
-- isOneElmt(lc) true jika list of integer lc hanya
-- mempunyai satu elemen
-- REALISASI
isOneElmt lc = (length lc) == 1
```

Apa yang sudah dipelajari?

- Rekursifitas dan analisis rekurens
- Memanfaatkan analisis rekurens untuk konstruksi program rekursif
- Perkenalan list sebagai struktur data rekursif

Bahan

- Diktat “Dasar Pemrograman, Bag. Pemrograman Fungsional” oleh Inggriani Liem, revisi Februari 2014