IF1210 Dasar Pemrograman

Ekspresi Lambda dan Fungsi sebagai Parameter Fungsi

Tim Pengajar IF1210

Sekolah Teknik Elektro dan Informatika

Tujuan

- Mahasiswa memahami kegunaan dari ekspresi lambda
- Berdasarkan pemahaman tersebut, mahasiswa mampu membuat fungsi sederhana yang menggunakan ekspresi lambda
- Mahasiswa mampu mengimplementasi ekspresi lambda dalam Haskell

Ekspresi Lambda

Ekspresi Lambda

- Notasi lambda memungkinkan penggunaan fungsi tanpa harus diberi nama
- Contoh ekspresi lambda (notasi fungsional):
  - $\lambda x.x \rightarrow (\lambda x.x)\ 3 = 3$
  - $\lambda x.x^3 \rightarrow (\lambda x.x^3)\ 3 = 27$
  - $\lambda x.x+1 \rightarrow (\lambda x.x+1)\ 3 = 4$
  - $\lambda x.x+4 \rightarrow (\lambda x.x+4)\ 3 = 7$
  - $\lambda x.1/((x)*(x+2))$
    $\rightarrow (\lambda x.1/((x)*(x+2)))\ 3 = 6.666666666666667e\text{-}2$

Ekspresi Lambda dengan Lebih dari Satu Parameter

Contoh, untuk membuat ekspresi lambda yang menjumlahkan dua bilangan:

$\lambda x,y.x+y$ ATAU $\lambda x.\lambda y.x+y$

Sehingga:

$(\lambda x,y.x+y)\ (2,3) = (\lambda x.\lambda y.x+y)\ (2,3)$

$= (\lambda y.2+y)\ (3)$

$= 2 + 3 = 5$

Notasi Haskell

Notasi Fungsional	Notasi Haskell
λ	\
.	->

Contoh

Notasi Fungsional	Notasi Haskell
λx.x	(\x -> x)
$\lambda x.x^3$	(\x -> x*x*x)
λx.x+1	(\x -> x + 1)
λx.x+4	(\x -> x + 4)
λx.1/((x)*(x+2))	(\x -> 1/((x)*(x+2)))

Contoh Aplikasi

```
> (\x -> x) 5
5
> (\x -> x+1) 5
6
> (\x -> x+4) 5
9
> (\x -> x*x*x) 5
125
> (\x -> 1/((x)*(x+2))) 5
2.857142857142857e-2
```

Ekspresi Lambda dengan Lebih dari Satu Parameter

λx,y.2*x+y Haskell: \(x,y)-> 2*x+y

ATAU

λx.λy.2*x+y Haskell: \x -> (\y -> 2*x+y)

Aplikasi di Haskell:

```
> (\(x,y)->2*x+y) (2,3)
7
```

ATAU

```
> (\x -> (\y -> 2*x+y) 3) 2
7
```

Aspek Fungsi sebagai Parameter Fungsi

Latihan

1. Diberikan ADT List seperti terdefinisi pada saat perkuliahan dengan primitif dasar: **konso**, **konsDot**, **head**, **tail**, **last**, **init**, **isEmpty**, dan **isOneElmt**.

   Tuliskan definisi, spesifikasi, dan realisasi dari fungsi CountEven yang menerima sebuah list of integer dan mengembalikan banyaknya elemen list yang merupakan bilangan genap.

   CountEven [25,**40**,**26**,**0**,13,15,97,**88**] = 4

   CountEven [25,13,15,97] = 0

   CountEven [ ] = 0

Latihan

2. Diberikan ADT List seperti terdefinisi pada saat perkuliahan dengan primitif dasar: **konso**, **konsDot**, **head**, **tail**, **last**, **init**, **isEmpty**, dan **isOneElmt**.

   Tuliskan definisi, spesifikasi, dan realisasi dari fungsi Count0 yang menerima sebuah list of integer dan mengembalikan banyaknya kemunculan angka 0 pada list.

   Count0 [12,**0**,89,41,**0**,23,**0**,**0**,**0**,18,**0**,15] = 6

   Count0 [12,89,41,23,18,15] = 0

   Count0 [ ] = 0

Latihan

3. Diberikan ADT List seperti terdefinisi pada saat perkuliahan dengan primitif dasar: **konso**, **konsDot**, **head**, **tail**, **last**, **init**, **isEmpty**, dan **isOneElmt**.

   Tuliskan definisi, spesifikasi, dan realisasi dari fungsi CountMultOf5 yang menerima sebuah list of integer dan mengembalikan banyaknya elemen list yang merupakan kelipatan dari 5.

   CountMultOf5 [**5**,1,**20**,7,77,**45**,**0**,14,**15**] = 5

   CountMultOf5 [1,7,77,14] = 0

   CountMultOf5 [ ] = 0

Latihan

4. Diberikan ADT List seperti terdefinisi pada saat perkuliahan dengan primitif dasar: **konso**, **konsDot**, **head**, **tail**, **last**, **init**, **isEmpty**, dan **isOneElmt**.

   Tuliskan definisi, spesifikasi, dan realisasi dari fungsi CountCond yang menerima sebuah list of integer dan mengembalikan banyaknya elemen list yang memenuhi kondisi berikut:

   - Untuk bilangan ganjil: habis dibagi tiga tapi tidak habis dibagi 5
   - Untuk bilangan genap: bernilai antara 51 hingga 100
   - Sama dengan 0

   CountCond([**27**,15,40,**78**,**99**,**90**,**66**,45,**0**,**98**,2,30,**51**]) = 8
   CountCond([15,40,**90**,45,2,30]) = 1
   CountCond([ ]) = 0

Saat ini sebagian besar dari Anda pasti berpikiran seperti salah satu dari mahasiswa ini… ☺

Aduh.. Hari ini kok dosenku tidak kreatif sekali … Dari tadi latihannya hitung-hitung terus …

Pak/Bu… Ganti dong soalnya..

Aduuuhh… Seandainya ada copy/paste untuk tulisan tangan..

Bosen banget nulis programnya … Semua hampir sama

```
countEven ::  [Int] -> Int
{- CountEven(L) mengembalikan banyaknya
elemen list yang merupakan bilangan genap -}
countEven l =
    if isEmpty l then 0 -- basis
    else -- rekurens
        (if mod (head l) 2 == 0 then 1 else 0) +
        countEven (tail l)
```

```
countZero ::  [Int] -> Int
{- CountZero(L) mengembalikan banyaknya
elemen list yang merupakan bilangan 0 -}
countZero l =
    if isEmpty l then 0 -- basis 0
    else -- rekurens
        (if (head l) == 0 then 1 else 0) +
        countZero (tail l)
```

```
countMultOf5 ::  [Int] -> Int
{- CountMultOf5(L) mengembalikan banyaknya
elemen list yang merupakan kelipatan 5 -}
countMultOf5 l =
    if isEmpty l then 0 -- basis
    else -- rekurens
        (if mod (head l) 5 == 0 then 1 else 0) +
        countMultOf5 (tail l)
```

```
countCond ::  [Int] -> Int
{- CountCond(L) mengembalikan banyaknya
elemen list yang memenuhi kondisi tertentu -}
countCond l =
    if isEmpty l then 0 -- basis 0
    else -- rekurens
        (if ((mod (head l) 2)/=0 && (mod (head l)
            3)==0 && (mod (head l) 5)/=0) ||
           ((mod (head l) 2)==0 && (head l)>=51
            && (head l)<=100) || ((head l)==0)
         then 1 else 0) + countCond (tail l)
```

Hal seperti ini sering terjadi…

Misalnya, setelah selesai masa pendaftaran mahasiswa baru

Tolong hitung jumlah mahasiswa baru yang pria

Tolong hitung jumlah mahasiswa baru dari pulau Kalimantan

Tolong hitung jumlah mahasiswa baru dari jalur SNMPTN

Tolong hitung jumlah mahasiswa baru TPB STEI

Tolong hitung jumlah mahasiswa baru TPB STEI dari pulau Sumatera

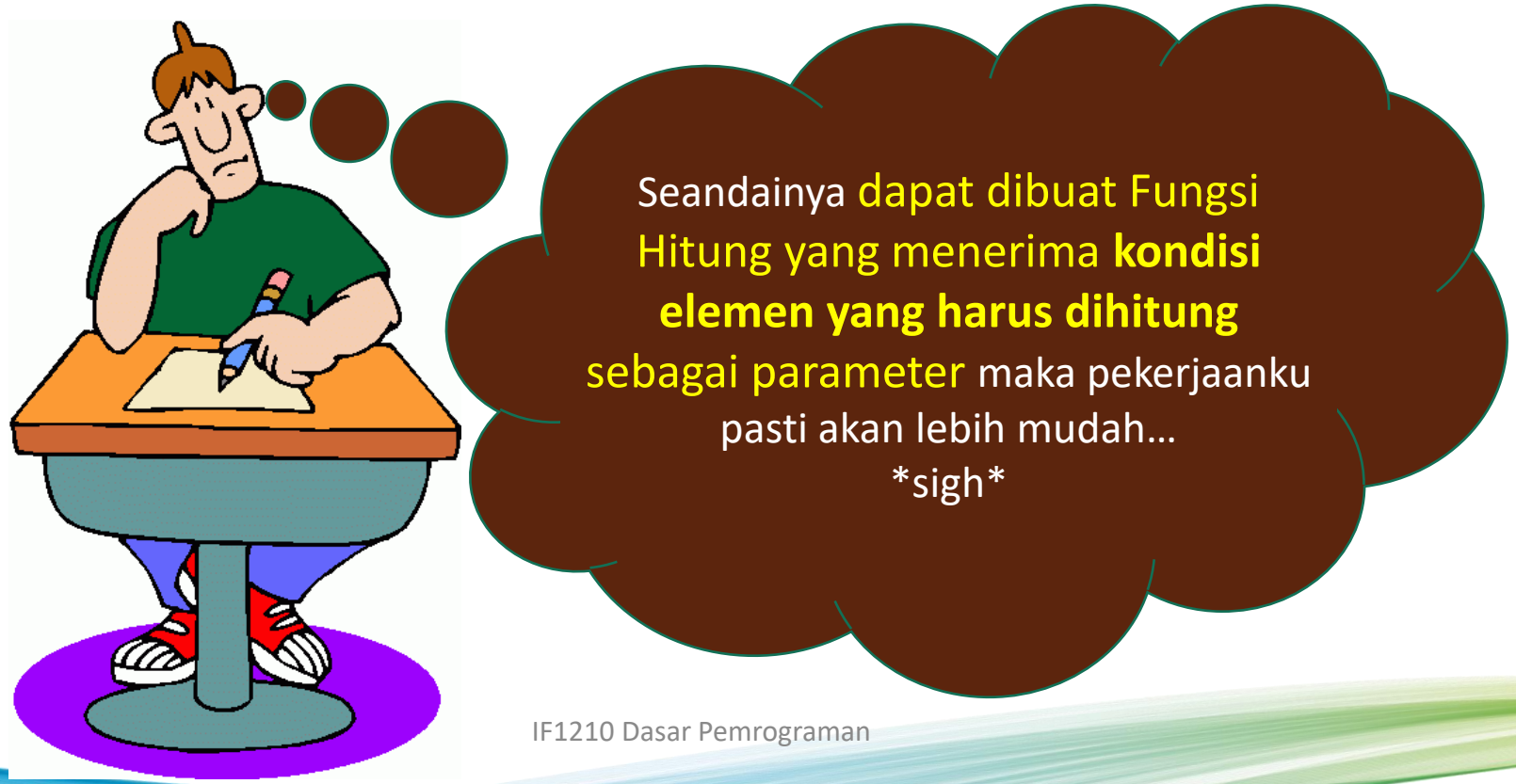
Tolong hitung jumlah mahasiswa baru perempuan

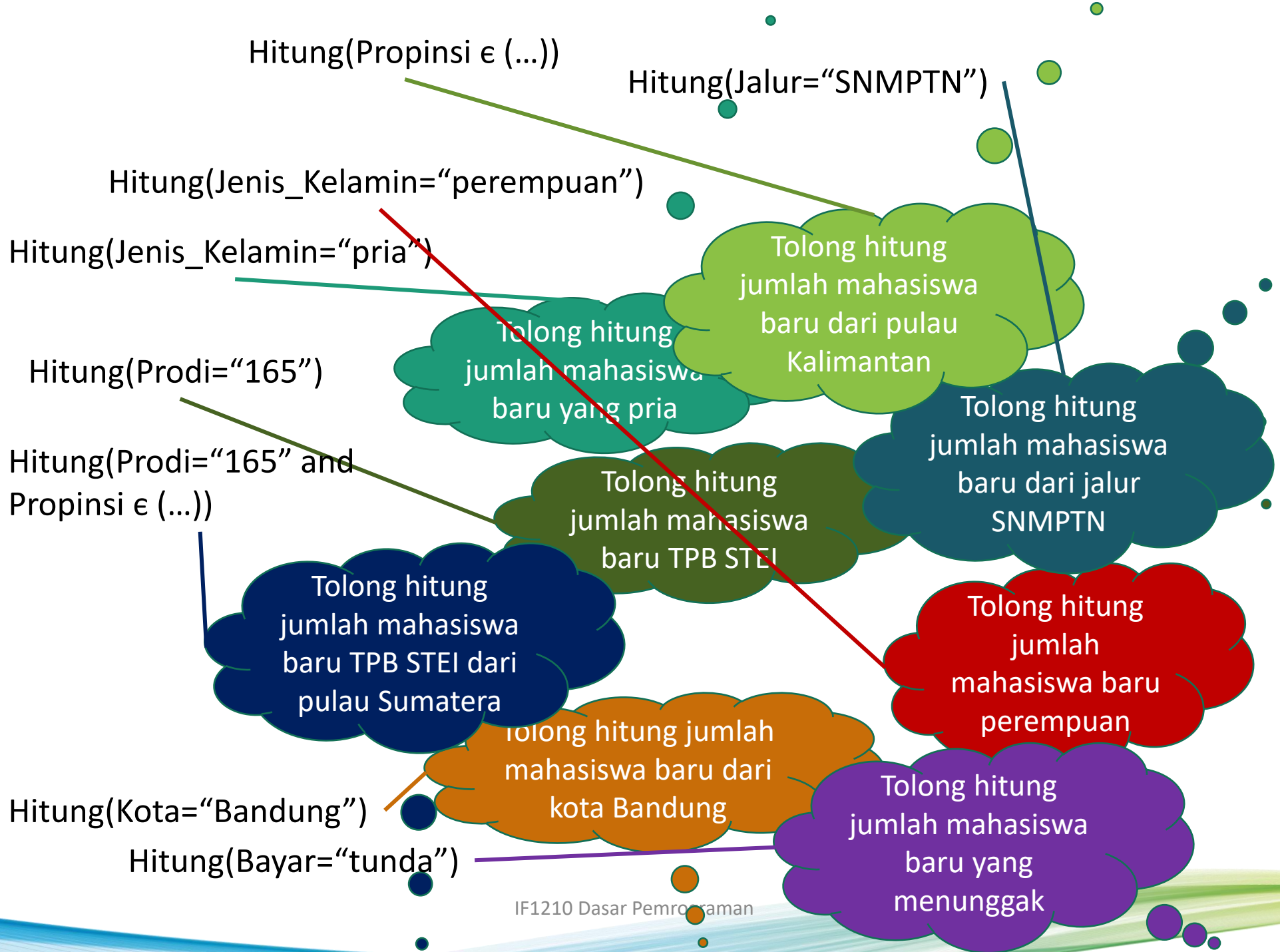
Tolong hitung jumlah mahasiswa baru dari kota Bandung

Tolong hitung jumlah mahasiswa baru yang menunggak

Jika Anda adalah Programmer di slide sebelumnya...

Apa yang terpikir oleh Anda pada saat menghadapi keadaan seperti itu?




Hitung(Propinsi ϵ (...))
Hitung(Jalur="SNMPTN")
Hitung(Jenis_Kelamin="perempuan")
Hitung(Jenis_Kelamin="pria")
Hitung(Prodi="165")
Hitung(Prodi="165" and Propinsi ϵ (...))
Hitung(Kota="Bandung")
Hitung(Bayar="tunda")
Tolong hitung jumlah mahasiswa baru dari pulau Kalimantan
Tolong hitung jumlah mahasiswa baru yang pria
Tolong hitung jumlah mahasiswa baru dari jalur SNMPTN
Tolong hitung jumlah mahasiswa baru TPB STEI
Tolong hitung jumlah mahasiswa baru TPB STEI dari pulau Sumatera
Tolong hitung jumlah mahasiswa baru perempuan
Tolong hitung jumlah mahasiswa baru dari kota Bandung
Tolong hitung jumlah mahasiswa baru yang menunggak

Fungsi Sebagai Parameter Fungsi

- Kondisi sebagai parameter fungsi diimplementasikan dalam bentuk fungsi.
- Oleh sebab itu, diperkenalkan istilah Fungsi sebagai Parameter dari Fungsi.
- Kemampuan mendefinisikan fungsi sebagai parameter fungsi ini merupakan bentuk **abstraksi** yang lebih tinggi, karena sebagian ekspresi dapat ditunda pendefinisiannya hingga pada saat fungsi dipanggil.

Perhatikan fungsi-fungsi counting berikut.

Basis sama

```
countEven ::  [Int] -> Int
{- CountEven(L) mengembalikan banyaknya
elemen list yang merupakan bilangan genap -}
countEven l =
    if isEmpty l then 0 -- basis
    else -- rekurens
        (if mod (head l) 2 == 0 then 1 else 0) +
        countEven (tail l)
```

```
countZero ::  [Int] -> Int
{- CountZero(L) mengembalikan banyaknya
elemen list yang merupakan bilangan 0 -}
countZero l =
    if isEmpty l then 0 -- basis
    else -- rekurens
        (if (head l) == 0 then 1 else 0) +
        countZero (tail l)
```

```
countMultOf5 ::  [Int] -> Int
{- CountMultOf5(L) mengembalikan banyaknya
elemen list yang merupakan kelipatan 5 -}
countMultOf5 l =
    if isEmpty l then 0 -- basis
    else -- rekurens
        (if mod (head l) 5 == 0 then 1 else 0) +
        countMultOf5 (tail l)
```

```
countCond ::  [Int] -> Int
{- CountCond(L) mengembalikan banyaknya
elemen list yang memenuhi kondisi tertentu -}
countCond l =
    if isEmpty l then 0 -- basis
    else -- rekurens
        (if ((mod (head l) 2)/=0 && (mod (head l)
              3)==0 && (mod (head l) 5)/=0) ||
             ((mod (head l) 2)==0 && (head l)>=51
              && (head l)<=100) || ((head l)==0)
         then 1 else 0) + countCond (tail l)
```

Perhatikan fungsi-fungsi counting berikut.

countEven :: [Int] -> Int
{- CountEven(L) mengembalikan banyaknya elemen list yang merupakan bilangan genap -}
countEven l =
    if isEmpty l then 0 -- basis
    else -- rekurens
        (if mod (head l) 2 == 0 then 1 else 0) + countEven (tail l)

countZero :: [Int] -> Int
{- Cou
eleme
count
    if isEmpty l then 0 -- basis
    else -- rekurens
        (if (head l) == 0 then 1 else 0) + countZero (tail l)

Pola rekurens sama:
(if (**<condition>**) then 1 else 0) + **<function>**(tail l)

countMultOf5 :: [Int] -> Int
{- CountMultOf5(L) mengembalikan banyaknya elemen list yang merupakan kelipatan 5 -}
countMultOf5 l =
    if isEmpty l then 0 -- basis
    else -- rekurens
        (if mod (head l) 5 == 0 then 1 else 0) + countMultOf5 (tail l)

countCond :: [Int] -> Int
{- CountCond(L) mengembalikan banyaknya elemen list yang memenuhi kondisi tertentu -}
countCond l =
    if isEmpty l then 0 -- basis
    else -- rekurens
        (if ((mod (head l) 2)/=0 && (mod (head l) 3)==0 && (mod (head l) 5)/=0) || ((mod (head l) 2)==0 && (head l)>=51 && (head l)<=100) || ((head l)==0) then 1 else 0) + countCond (tail l)

Perhatikan fungsi-fungsi counting berikut.

Yang membedakan adalah kondisi pada bagian rekurens

Bagaimana jika kondisi ini menjadi **parameter** fungsi?

```
countEven :: [Int] -> Int
{- CountEven(L) mengembalikan banyaknya
elemen list yang merupakan bilangan genap -}
countEven l =
    if isEmpty l then 0 -- basis
    else -- rekurens
        (if mod (head l) 2 == 0 then 1 else 0) +
        countEven (tail l)
```

```
co
{-
el
co
    (if (head l) == 0 then 1 else 0) +
    countZero (tail l)
```

```
countMultOf5 :: [Int] -> Int
{- CountMultOf5(L) mengembalikan banyaknya
elemen list yang merupakan kelipatan 5 -}
countMultOf5 l =
    if isEmpty l then 0 -- basis
    else -- rekurens
        (if mod (head l) 5 == 0 then 1 else 0) +
        countMultOf5 (tail l)
```

```
countCond :: [Int] -> Int
{- CountCond(L) mengembalikan banyaknya
elemen list yang memenuhi kondisi tertentu -}
countCond l =
    if isEmpty l then 0 -- basis
    else -- rekurens
        (if ((mod (head l) 2)/=0 && (mod (head l)
            3)==0 && (mod (head l) 5)/=0) ||
           ((mod (head l) 2)==0 && (head l)>=51
            && (head l)<=100) || ((head l)==0)
        then 1 else 0) + countCond (tail l)
```

Generalisasi Fungsi

```
countEven l =
    if isEmpty l then 0 -- basis
    else -- rekurens
        (if mod (head l) 2 == 0 then 1 else 0) +
        countEven (tail l)
```

```
countZero l =
    if isEmpty l then 0 -- basis
    else -- rekurens
        (if (head l) == 0 then 1 else 0) +
        countZero (tail l)
```

```
countMultOf5 l =
    if isEmpty l then 0 -- basis
    else -- rekurens
        (if mod (head l) 5 == 0 then 1 else 0) +
        countMultOf5 (tail l)
```

```
countCond l =
    if isEmpty l then 0 -- basis
    else -- rekurens
        (if ((mod (head l) 2)/=0 && (mod (head l)
            3)==0 && (mod (head l) 5)/=0) ||
           ((mod (head l) 2)==0 && (head l)>=51
            && (head l)<=100) || ((head l)==0)
         then 1 else 0) + countCond (tail l)
```

```
-- DEFINISI DAN SPESIFIKASI
isEven :: Int -> Bool
-- isEven n true jika n bil. Genap
isZero :: Int -> Bool
-- isZero n true jika n = 0
isMultOf5 :: Int -> Bool
-- isMultOf4 n true jika n kelipatan 5
isCond :: Int -> Bool
-- isCond n true jika n memenuhi kondisi…
```

```
-- REALISASI
isEven n = (mod n 2) == 0
isZero n = n == 0
isMultOf5 n = (mod n 5) == 0
isCond n =  ((mod n 2)/=0 && (mod n 3)==0
              && (mod n 5)/=0) ||
            ((mod n 2)==0 && n>=51
              && n<=100) ||
            (n==0)
```

Fungsi dengan Parameter Fungsi

- Definisikan fungsi **countIf** dengan masukan sebuah list of integer dan sebuah "fungsi" predikat yang menerima masukan sebuah integer dan menghasilkan nilai boolean

```
-- DEFINISI DAN SPESIFIKASI
countIf :: [Int] -> (Int -> Bool) -> Int
-- countIf li f menghasilkan banyaknya elemen list of integer
-- li yang memenuhi kondisi yang dinyatakan dengan fungsi f

-- REALISASI
countIf li f = if isEmpty li then 0   -- Basis
               else -- Rekurens
                   (if (f (head li)) then 1 else 0)
                     + (countIf (tail li) f)
```

Contoh Aplikasi

Asumsi: fungsi **isEven**, **isZero**, **isMultOf5**, dan **isCond** sudah terdefinisi dan terealisasi spt. pada slide 23

```
> countIf [25,40,26,0,13,15,97,88] isEven
4
> countIf [12,89,41,23,18,15] isZero
0
> countIf [] isMultOf5
0
> countIf [5,1,20,7,77,45,0,14,15] isMultOf5
5
> countIf [15,40,90,45,2,30] isCond
1
```

Aplikasi dengan Ekspresi Lambda

- Konstanta hasil fungsi dapat digunakan sebagai parameter efektif pada ekspresi fungsional
- Contoh ekspresi lambda untuk 4 fungsi sebelumnya

fungsi	Eksp. Lambda di Notasi Fungsional	Eksp. Lambda di Haskell
isEven	λx.x mod 2 = 0	\x-> (mod x 2) == 0
isZero	λx.x = 0	\x->x == 0
isMultOf5	λx.x mod 5 = 0	\x-> (mod x 5) == 0
isCond	λx.((x mod 2 ≠ 0) and (x mod 3 = 0) and (x mod 5 ≠ 0)) or ((x mod 2 = 0) and n ≥ 51 and n ≤ 100) or (n = 0)	\x-> (((mod x 2) /= 0) && ((mod x 3) == 0) && ((mod x 5) /= 0)) || (((mod x 2) == 0) && (n >= 51) && (n <= 100)) || (n == 0)

Contoh Aplikasi dengan Ekspresi Lambda

```
> countIf [25,40,26,0,13,15,97,88] (\x -> (mod x 2) == 0)
4
> countIf [12,89,41,23,18,15] (\x -> x == 0)
0
> countIf [] (\x -> (mod x 5) == 0)
0
> countIf [5,1,20,7,77,45,0,14,15] (\x -> (mod x 5) == 0)
5
> countIf [15,40,90,45,2,30] (\x -> (((mod x 2) /= 0) && ((mod x
  3) == 0) && ((mod x 5) /= 0)) || (((mod x 2) == 0) && (n >= 51)
  && (n <= 100)) || (n == 0))
1
```

Contoh-2: Offset List

- Tuliskan definisi, spesifikasi, dan realisasi sebuah fungsi yang melakukan “offset” atau perubahan nilai terhadap elemen list dan menghasilkan list baru dengan elemen hasil offset.
- Contoh: diberikan sebuah list of integer
  - Dengan fungsi offset **plus2**, akan menghasilkan list baru dengan nilai setiap elemen yang sudah bertambah 2
  - Dengan fungsi offset **minus1**, akan menghasilkan list baru dengan nilai setiap elemen yang sudah berkurang 1
  - Dengan fungsi offset **offKond**, akan menghasilkan list baru dengan nilai setiap elemen yang diubah sesuai ketentuan range tertentu

offsetList – Definisi, Spesifikasi, Realisasi

```
-- DEFINISI DAN SPESIFIKASI
offsetList :: [Int] -> (Int->Int) -> [Int]
-- offsetList li offset dengan li adalah list integer dan
-- offset adalah sebuah fungsi dengan definisi:
--                 offset i melakukan offset terhadap nilai i
-- offsetList menghasilkan sebuah list integer dengan semua elemen
-- sudah di-offset sesuai fungsi offset

-- REALISASI
offsetList li offset =
     if isEmpty li then []   -- Basis
     else  -- Rekurens
         konso (offset (head li)) (offsetList (tail li) offset)
```

offsetList – Fungsi offset

```
-- DEFINISI DAN SPESIFIKASI
plus2 :: Int -> Int
-- plus2 i menghasilkan i+2
minus1 :: Int -> Int
-- minus1 i menghasilkan i-1
offKond :: Int -> Int
-- offKond i menghasilkan i yang di-offset sesuai aturan...

-- REALISASI
plus2 i = i + 2
minus1 i = i - 1
offKond i
    | i>=0  && i<=40 = 10
    | i>=41 && i<=60 = 5
    | i>=61 && i<=89 = 3
    | i>89           = 1
    | otherwise      = 0
```

offsetList – Contoh Aplikasi dengan Parameter Fungsi Bernama

```
> offsetList [1,1,1,1,1] plus2
[3,3,3,3,3]

> offsetList [1,1,1,1,1] minus1
[0,0,0,0,0]

> offsetList [55,23,0,1,76] offKond
[5,10,10,10,3]
```

offsetList – Contoh Aplikasi dengan parameter ekspresi lambda

```
> offsetList [1,2,3,4,5] (\x->x+2)  -- plus2
[3,4,5,6,7]

> offsetList [1,2,3,4,5] (\x->x-1)  -- minus1
[0,1,2,3,4]

> offsetList [55,23,0,1,76] (\x -> if(x>=0 && x<=40) then
  10 else if (x>=41 && x<= 60) then 5 else if (x>=61 &&
  x<=89) then 3 else if (x>89) then 1 else 0)  -- offKond
[5,10,10,10,3]
```

Contoh-3: sigI, sigI3, sp8

Definisi deret sigI:

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

```
-- DEFINISI DAN SPESIFIKASI
sigI :: Int -> Int -> Int
{- sigI a b adalah fungsi untuk menghitung sigma(i)
   untuk nilai i pada interval a dan b sbb.:
   a+(a+1)+(a+1+1)+...+b,
   atau 0 jika interval "kosong" -}
```

```
-- REALISASI
sigI a b = if a > b then 0 -- Basis
           else -- Rekurens
               a + sigI (a+1) b
```

Contoh-3: sigI, sigI3, sp8

Definisi deret sigI:

sigI3 = $\sum_{i=a}^{b} i^3$

```
-- DEFINISI DAN SPESIFIKASI
sigI3 :: Int -> Int -> Int
{- sigI3 a b adalah fungsi untuk menghitung
   Sigma(i^3) untuk nilai i pada interval a dan b
   sbb.: a^3+(a+1)^3+(a+1+1)^3+...+b^3,
   atau 0 jika interval "kosong" -}
```

```
-- REALISASI
sigI3 a b = if a > b then 0 -- Basis
            else -- Rekurens
                a*a*a + sigI3 (a+1) b
```

Contoh-3: sigl, sigl3, sp8

Definisi deret sigl:

$$\text{sp8} = \sum_{i=a}^{b} \frac{1}{i*(i+2)}$$

```
-- DEFINISI DAN SPESIFIKASI
sp8 :: Int -> Int -> Int
{- 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 0 -- Basis 0
          else -- Rekurens
              (fromIntegral 1 /
               fromIntegral (a * (a+2))) + sp8 (a+4) b
```

fromIntegral digunakan untuk mengubah tipe x dari Int ke Float agar sesuai dengan definisi

Ilustrasi Program Fungsional (Cont.)

Fungsi sigI, SigI3, dan SP8 memiliki kemiripan

```
-- REALISASI
sigI a b = if a > b then 0 -- Basis
           else -- Rekurens
              a + sigI (a+1) b
```

```
-- REALISASI
sigI3 a b = if a > b then 0 -- Basis
            else -- Rekurens
               a*a*a + sigI3 (a+1) b
```

```
-- REALISASI
sp8 a b =
   if a > b then 0 -- Basis
   else -- Rekurens
      (fromIntegral 1) /
      (fromIntegral (a * (a+2))) +
                    sp8 (a+4) b)
```

Ketiga fungsi menghitung sigma dari elemen suatu deret antara dua bilangan

Ketiga fungsi memiliki basis yang sama

Ketiga fungsi memiliki pola ekspresi yang serupa pada bagian rekurens:
<aplikasi suatu fungsi terhadap parameter pertama> +
<pemanggilan rekursif dengan parameter pertama menuju basis>

Generalisasi Fungsi

```
-- REALISASI
sigI a b =
   if a > b then 0 -- Basis
   else -- Rekurens
     a + sigI (a+1) b
```

```
REALISASI
sigI3 a b =
   if a > b then 0 -- Basis
   else -- Rekurens
     a*a*a + sigI3 (a+1) b
```

```
-- REALISASI
sp8 a b =
   if a > b then 0 -- Basis
   else -- Rekurens
     (fromIntegral 1 /
      fromIntegral (a * (a+2))) +
                          sp8 (a+4) b
```

```
DEFINISI DAN SPESIFIKASI
id :: Int -> Int
-- id i mengirimkan nilai i
p1 :: Int -> Int
-- p1 i mengirimkan nilai i+1
p4 :: Int -> Int
-- p4 i mengirimkan nilai i+4
cube :: Int -> Int
-- cube i mengirimkan nilai i^3
t :: Int -> Float
-- t i mengirimkan nilai 1/(i*(i+2))
```

```
-- REALISASI
id i = i
p1 i = i+1
p4 i = i+4
cube i = i*i*i
t i = fromIntegral 1 /
        fromIntegral ((i)*(i+2))
```

Generalisasi Fungsi

- Fungsi **id**, **p1**, **p4**, **cube** mengembalikan nilai integer, tetapi fungsi **t** mengembalikan nilai real (float)
  - Untuk menggunakan semua fungsi sebagai parameter, perlu didefinisikan type numerik: gabungan dari type integer dan real
- Definisikan "Sigma" dari deret: sebagai rumus/fungsi umum dari penjumlahan suku deret dengan **fungsi sebagai parameter fungsi**

Fungsi Sigma (notasi fungsional)

DEFINISI DAN SPESIFIKASI

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) = SigI(a,b)
- Sigma(a,b,Cube,P1) = SigI3(a,b)
- Sigma(a,b,T,P4) = SP8(a,b)

Sigma - Notasi Haskell

Definisi type numerik membutuhkan penanganan khusus di Haskell → tidak dibahas di kuliah ini

Berikut contoh implementasi fungsi sigma di Haskell:

```
-- DEFINISI DAN SPESIFIKASI
sigma::Int->Int->(Int->Float)->(Int->Int)->Float
{- Sigma (a,b,f,s) adalah penjumlahan dari deret/series
   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 0  -- Basis
                else -- Rekurens
                    (f a) + sigma (s a) b f s
```

Ekspresi Lambda

- Contoh ekspresi lambda untuk beberapa fungsi sebelumnya:
  - Id : $\lambda x.x$
  - Cube : $\lambda x.x^3$
  - P1 : $\lambda x.x+1$
  - P4 : $\lambda x.x+4$
  - T : $\lambda x.1/((x)*(x+2))$
- Maka, fungsi SigI, SigI3, SP8 dapat diperoleh melalui aplikasi fungsi Sigma:
  - SigI(a,b) ⟹ Sigma(a, b, $\lambda x.x$, $\lambda x.x+1$)
  - SigI3(a,b) ⟹ Sigma(a, b, $\lambda x.x^3$, $\lambda x.x+1$)
  - SP8(a,b) ⟹ Sigma(a, b, $\lambda x.1/((x)*(x+2))$,$\lambda x.x+4$)

Sigma - Contoh Aplikasi

```
> sigma 2 5 (\x->fromIntegral x) (\x->x+1)
14.0

> sigma 2 5 (\x->fromIntegral x*fromIntegral x*fromIntegral x) (\x->x+1)
224.0

> sigma 2 5 (\x->1/((fromIntegral x)*(fromIntegral x+2))) (\x->x+4)
0.125
```

fromIntegral digunakan untuk mengubah tipe x dari Int ke Float agar sesuai dengan definisi

Apakah Ekspresi Lambda dapat berupa ekspresi rekursif?

- Ekspresi rekursif mengandung aplikasi fungsi yang memuat ekspresi tersebut.
- Dalam pemakaian ekspresi lambda, ekspresi dituliskan secara langsung (bukan sebagai bagian dari fungsi)

→ekspresi lambda tidak bisa mengandung rekursif

Latihan: Filter List

- Tuliskan definisi, spesifikasi, dan realisasi sebuah fungsi yang melakukan "filter" atau penyaringan terhadap elemen list dan menghasilkan list baru dengan elemen yang lolos kriteria filter.
- Contoh: diberikan sebuah list integer
  - Dengan fungsi filter **isPos**, akan menghasilkan list baru yang hanya berisi elemen list masukan yang positif
  - Dengan fungsi filter **isNeg**, akan menghasilkan list baru yang hanya berisi elemen list masukan yang negatif
  - Dengan fungsi filter **isKabisat**, akan menghasilkan list baru yang hanya berisi elemen list yang masuk kategori tahun kabisat
- Buatlah pula contoh aplikasi dengan menggunakan fungsi-fungsi di atas dan contoh aplikasi dengan menggunakan ekspresi Lambda

Apa yang sudah diperoleh hari ini?

- Memahami kegunaan dari ekspresi lambda
- Berdasarkan pemahaman tersebut, mampu membuat fungsi sederhana yang menggunakan ekspresi lambda
- Mampu mengimplementasi ekspresi lambda dalam Haskell

Bahan

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