IF1210 Dasar Pemrograman

Berpikir Abstrak dalam Paradigma Pemrograman Fungsional

Tim Pengajar IF1210

Sekolah Teknik Elektro dan Informatika

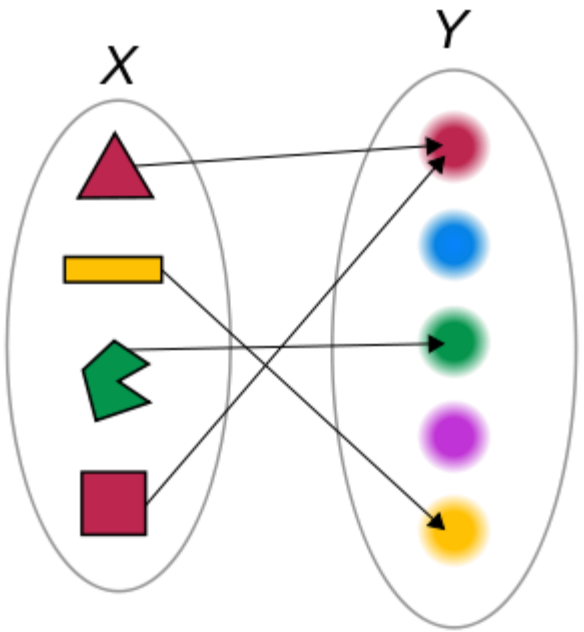
Sebelum dimulai…

- Sumber materi Konsep pemrograman Fungsional “Diktat Dasar Pemrograman Bagian Pemrograman Fungsional”
  - Disediakan di Edunex
  - Notasi fungsional kadang masih digunakan → tersedia dalam diktat
- Secara umum, materi pemrograman fungsional menggunakan Bahasa Haskell
  - Sumber belajar Bahasa Haskell: https://www.haskell.org/
  - Disediakan slides untuk belajar di Edunex

- *Important note:*
  - Banyak cara untuk menulis program di Haskell untuk menyelesaikan suatu persoalan
  - Di kuliah ini yang digunakan dan diterima hanyalah cara yang diajarkan di kelas → pelajari dengan sebaik-baiknya

Paradigma Fungsional

- Didasari oleh konsep pemetaan dan fungsi di matematika
- Pemrogram mengasumsikan bahwa ada fungsi-fungsi yang dapat dilakukan → penyelesaian masalah didasari atas aplikasi dari fungsi-fungsi
- Kelakuan program adalah suatu rantai transformasi dari sebuah keadaan awal menuju ke suatu keadaan akhir, yang mungkin melalui keadaan antara



Paradigma Fungsional (cont.)

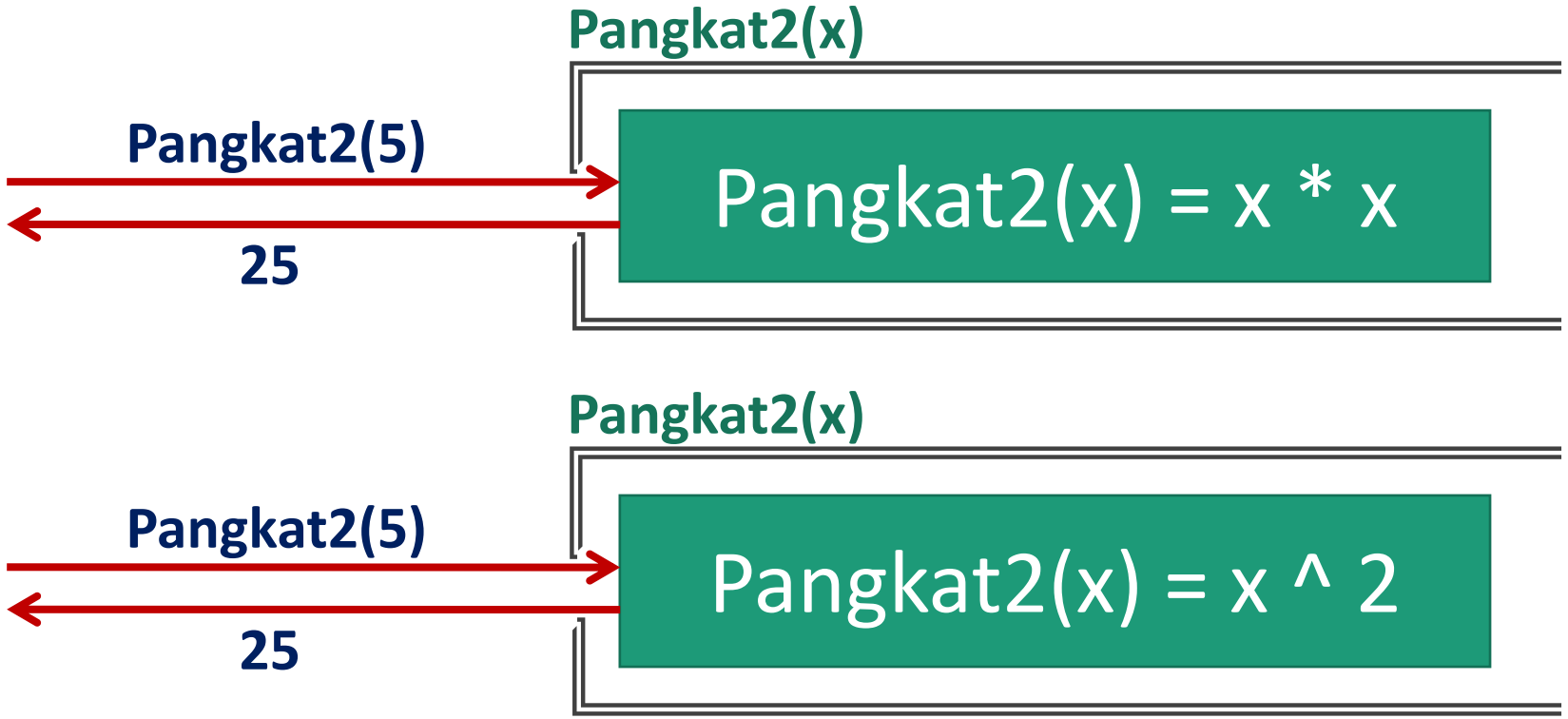
- Pada hakekatnya program dibuat untuk melaksanakan suatu fungsi tertentu sesuai dengan kebutuhan pemakai
- Paradigma fungsional memiliki pendekatan berpikir melalui fungsi (apa yang akan direalisasikan) dan “bebas memori” serta tidak mementingkan sekuens/urutan instruksi
- Programmer tidak perlu mengetahui bagaimana mesin melakukan eksekusi

Paradigma Fungsional (cont.)

- Didasari pada konsep pemetaan dan fungsi pada matematika
- Fungsi : asosiasi (pemetaan) antara 2 type yaitu domain dan range, yang dapat berupa:
  - Type dasar
  - Type terkomposisi (bentukan)
- Fungsi seperti “kotak hitam” (black box) → abstraksi

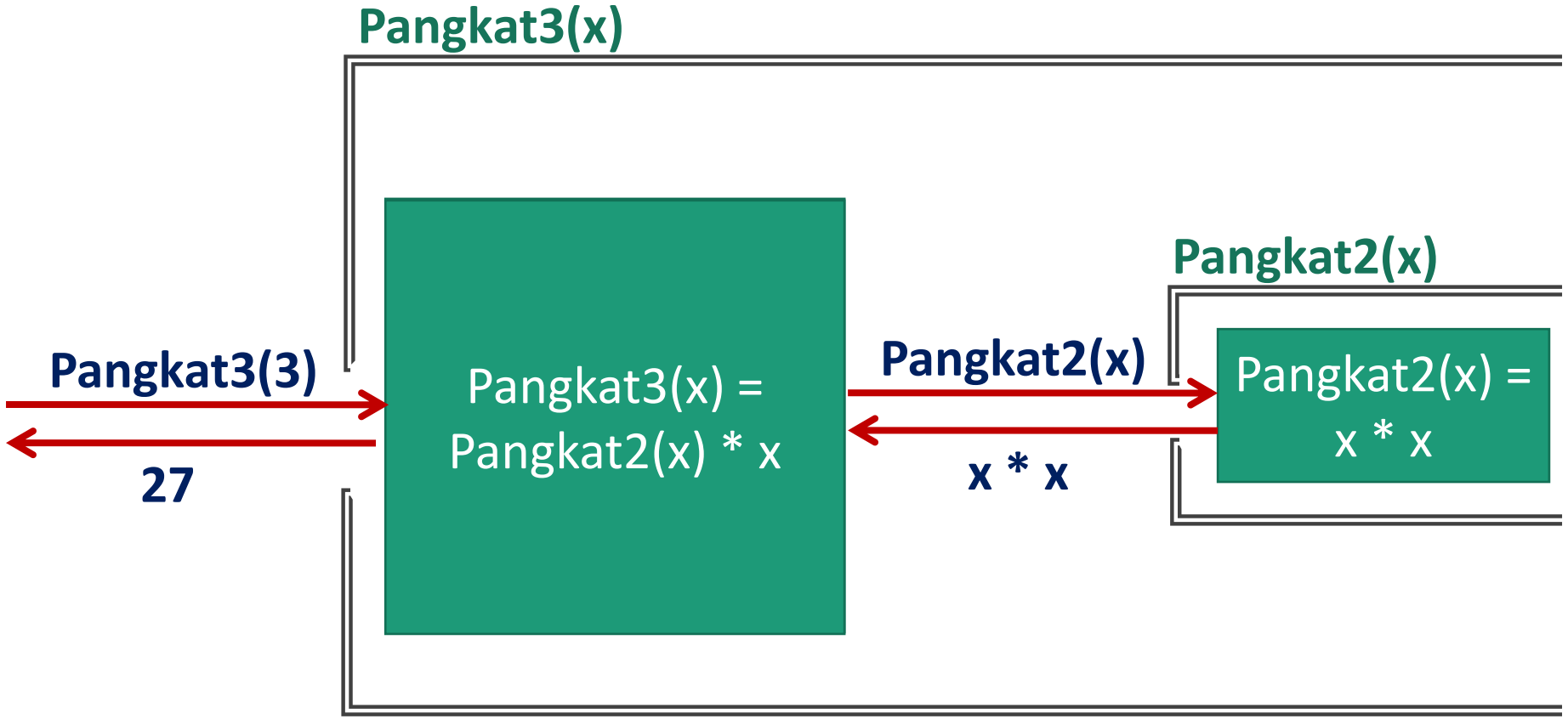
- Tujuan memrogram fungsional: merakit isi kotak hitam

Fungsi Sebagai Bentuk Abstraksi



Pengguna fungsi tidak perlu mengetahui bagaimana fungsi diimplementasikan.
Perubahan implementasi tidak mempengaruhi cara berkomunikasi dengan fungsi.

Fungsi Sebagai Bentuk Abstraksi (cont.)



Sebuah fungsi dapat di-'rakit' dengan memanfaatkan fungsi-fungsi lainnya

Operator

- Komputer mempunyai ALU (Arithmetic dan Logic Unit), oleh sebab itu mampu melakukan perhitungan numerik dan operasi logic
- Operator adalah “sesuatu” paling dasar untuk mengoperasikan suatu nilai bertype tertentu
- Contoh dalam fungsi pangkat3=x*x*x, operatornya adalah “*”
- Type yang dapat dioperasikan oleh operator adalah type dasar (misal: numerik <integer, real>, character)
- Akan dibahas lebih lanjut

Ekspresi

- Ekspresi : gabungan operan dan operator.
- Operan dapat berupa suatu nilai yang bertype sesuai operator, atau hasil aplikasi fungsi
- Contoh ekspresi x*x*x
- Ekspresi fungsional :
  - Ekspresi aritmatika, logika
  - Ekspresi kondisional
  - Ekspresi rekursif

Ekspresi Fungsional

- Program fungsional direalisasikan dalam bentuk ekspresi fungsional
- Ekspresi: sebuah teks yang terdiri dari: nama, simbol, operator/fungsi, ( ), yang dapat menghasilkan suatu nilai berkat evaluasi dari ekspresi
- Hasil evaluasi (perhitungan) suatu ekspresi dasar dapat berupa nilai numerik atau boolean.

Operator dan Ekspresi

- Fungsi paling dasar: **operator**
- Operator aritmatika: *, /, +, -
  - Contoh ekspresi aritmatika: (3+4)*5
  - Hasil evaluasi: 35
- Dalam Haskell mod dan div adalah suatu fungsi
- Operator relasional: <, >, =, ≤, ≥ , ≠
  - Dalam Haskell: <, >, ==, <=, >=, /=
- Operator boolean: and , or , not
  - Dalam Haskell: &&, ||, not
  - Contoh ekspresi boolean: not ((3<5) and (4≥6))
  - Contoh ekspresi dalam Haskell: not ((3<5) && (4>=6))
    Hasil evaluasi: True

Penulisan Ekspresi

- Penulisan ekspresi: infix, prefix,postfix
- Notasi fungsional: penulisan infix
- Notasi Haskell: penulisan infix

Jenis	Ekspresi Aritmatika	Ekspresi Boolean
Infix	(3+4)*5	3 < 5
Prefix	(* (+ 3 4) 5)	< 3 5
Postfix	(3 4 +) 5 *	3 5 <

Type

- Domain dan range fungsi didefinisikan dalam bentuk 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, string, boolean
  - Dalam Haskell: Int, Float, Char, String, Bool
  - Type bentukan → dibuat sendiri (untuk kuliah paradigma fungsional, tipe bentukan tidak diberikan)

Notasi Fungsional

- **Definisi Fungsi**
  - Memberikan identitas fungsi
  - Nama fungsi, Domain (parameter input), Range (definisi hasil)
  - Cth: fungsi Pangkat3, domain integer, range integer
    `Pangkat3: integer → integer`
- **Spesifikasi**
  - Apa yang akan dikerjakan oleh fungsi
  - Cth: `Fungsi Pangkat3(x) berarti menghitung pangkat tiga dari nilai x`
- **Realisasi**
  - Ekspresi fungsional, "bagaimana" program direalisasi menjadi instruksi komputer
  - Cth: penghitungannya: x * x * x ➔ `Pangkat3(x) : x * x * x`
- **Aplikasi**
  - Pemakaian fungsi yang sudah terdefinisi
  - Cth: `Pangkat3(3) + Pangkat3(-5)`

Notasi Haskell

- **Definisi Fungsi**
  - Memberikan identitas fungsi
  - Nama fungsi, Domain (parameter input), Range (definisi hasil)
  - Cth: fungsi Pangkat3, domain integer, range integer

    ```
    Pangkat3 :: Int -> Int
    ```
- **Spesifikasi**
  - Apa yang akan dikerjakan oleh fungsi
  - Contoh:

    ```
    Fungsi Pangkat3 x berarti menghitung pangkat tiga dari nilai x
    ```
- **Realisasi**
  - Ekspresi fungsional, "bagaimana" program direalisasi menjadi instruksi komputer
  - Cth: penghitungannya: x * x * x ➔ `Pangkat3 x = x * x * x`
- **Aplikasi**
  - Pemakaian fungsi yang sudah terdefinisi
  - Cth: `Pangkat3 3 + Pangkat3 (-5)`

Template Notasi Fungsional

JUDUL	**Nama-Fungsi (list-parameter-formal)**
DEFINISI DAN SPESIFIKASI Nama-Fungsi : domain → range { Tuliskan spesifikasi fungsi dengan nama, domain, dan range yang disebutkan di atas. }	
REALISASI Nama-Fungsi (list-parameter) : <ekspresi-fungsional>	
APLIKASI ⇒Nama-Fungsi (list-parameter-aktual) ⇒Nama-Fungsi (list-parameter-aktual) ⇒Nama-Fungsi (list-parameter-aktual)	

Bagian yang berkomunikasi dengan dunia luar

JUDUL **Nama-Fungsi (list-parameter-formal)**

DEFINISI DAN SPESIFIKASI

Nama-Fungsi : domain → range

{ Tuliskan spesifikasi fungsi dengan nama, domain, dan range yang disebutkan di atas. }

REALISASI

Nama-Fungsi (list-parameter) : <ekspresi-fungsional>

APLIKASI

⇒Nama-Fungsi (list-parameter-aktual)

⇒Nama-Fungsi (list-parameter-aktual)

⇒Nama-Fungsi (list-parameter-aktual)

Bagian internal fungsi, tidak perlu diketahui pengguna (inside the wall)

Template Notasi Haskell

```
-- JUDUL                                  Nama-Fungsi (list-parameter-formal)

-- DEFINISI DAN SPESIFIKASI
<Nama-Fungsi> :: <domain> -> <range>
     -- Tuliskan spesifikasi fungsi dengan nama, domain, dan
     -- range yang disebutkan di atas.

-- REALISASI
<Nama-Fungsi> <list-parameter> = <ekspresi-fungsional>

-- APLIKASI
-- <Nama-Fungsi> <list-parameter-actual>
```

Bagian yang berkomunikasi dengan dunia luar

```
-- JUDUL                              Nama-Fungsi (list-parameter-formal)
-- DEFINISI DAN SPESIFIKASI
<Nama-Fungsi> :: <domain> -> <range>
     -- Tuliskan spesifikasi fungsi dengan nama, domain, dan
     -- range yang disebutkan di atas.
-- REALISASI
<Nama-Fungsi> <list-parameter> = <ekspresi-fungsional>
-- APLIKASI
-- <Nama-Fungsi>  <list-parameter-actual>
```

Bagian internal fungsi, tidak perlu diketahui pengguna (inside the wall)

Pangkat2 (FX2) - Notasi Fungsional

- Menghitung nilai pangkat dua dr sebuah bilangan bulat (integer)
- Contoh evaluasi (penggunaan) fungsi
  - jika x = 2 maka f(2) = 2 * 2 = 4
  - Jika x = 30 maka f(30) = 30 * 30 = 900
- → Ide dasar: f(x) = x * x
- Notasi Fungsional

PANGKAT2	**FX2(x)**
DEFINISI DAN SPESIFIKASI **FX2 : integer → integer** ***{ FX2(x) menghitung pangkat dua dari x, sebuah bilangan integer }***	
REALISASI **FX2 (x) : x * x**	
APLIKASI **→ FX2 (1)**	

Pangkat2 (fx2) - Notasi Haskell

```
-- PANGKAT2                                                    fx2 x
```

```
-- DEFINISI DAN SPESIFIKASI
fx2 ::  Int -> Int
    -- fx2 x menghitung pangkat dua dari x,
    -- x sebuah bilangan integer
```

```
-- REALISASI
fx2 x = x * x
```

```
-- APLIKASI
-- fx2 2
```

Pangkat3 (FX3) - Notasi Fungsional

- Menghitung nilai pangkat tiga dr sebuah bilangan bulat (integer)
- Contoh evaluasi (penggunaan) fungsi
  - jika x = 2 maka f(2) = 2 * 2 * 2 = 8
  - Jika x = 30 maka f(30) = 30 * 30 * 30 = 27000
- → Ide dasar: f(x) = x * x * x
- Notasi Fungsional

PANGKAT3	**FX3(x)**
DEFINISI DAN SPESIFIKASI **FX3: integer → integer** ***{ FX3(x) menghitung pangkat tiga dari x, sebuah bilangan integer }***	
REALISASI **FX3 (x) : x * x * x**	
APLIKASI **→ FX3 (1)**	

Pangkat3 (fx3) - Notasi Haskell

```
-- PANGKAT3                                                fx3 x
```

```
-- DEFINISI DAN SPESIFIKASI
fx3 ::  Int -> Int
   -- fx3 x menghitung pangkat tiga dari x,
   -- x sebuah bilangan integer
```

```
-- REALISASI
fx3 x = x * x * x
```

```
-- APLIKASI
-- fx3 2
```

Pangkat3 (FX3) versi 2

- Bagaimana jika kita manfaatkan fungsi FX2?
  - FX2 → FX2(x) = x * x
  - FX3 → FX3(x) = x * x * x
  - FX3 → FX3(x) = FX2(x) * x

FX2 disebut fungsi antara

Pangkat3 (fx3) versi 2 dalam Haskell

```
-- PANGKAT3 (versi 2)                                    fx3 x
```

```
-- DEFINISI DAN SPESIFIKASI
fx3 :: Int -> Int
 {- fx3(x) menghitung pangkat tiga dari x, sebuah bilangan
    integer dengan aplikasi fx2 sebagai fungsi antara -}
fx2 :: Int -> Int
 {- fx2(x)  menghitung pangkat dua dari x, sebuah bilangan
    integer -}
```

```
-- REALISASI
fx3 x =  (fx2 x) * x
fx2 x =  x * x
```

```
-- APLIKASI
-- fx3 5
```

Contoh Ekspresi Boolean/Predikat

- Output dari ekspresi aritmatika adalah angka hasil perhitungan,
  - Contoh: fx2(4) = 16
- Output dari ekspresi boolean?
  - Output adalah nilai benar (true) atau salah (false)
  - Nilai true atau false disebut nilai boolean
  - Contoh
    - Positif: IsPositif?
    - Apakah huruf A: IsAnA?
    - Apakah origin: IsOrigin?
    - Apakah valid: IsValid?

POSITIF → IsPositif

- Memeriksa apakah sebuah bilangan integer itu positif atau tidak
- Jika x bernilai lebih besar atau sama dengan 0 maka x adalah positif
- Berarti: fungsi isPositif benar jika nilai x lebih besar atau sama dgn 0
- Penulisan Dalam Notasi Haskell

```
-- POSITIF                              isPositif x
```

```
-- DEFINISI DAN SPESIFIKASI
isPositif :: Int -> Bool
  -- isPositif x benar jika x positif
```

```
-- REALISASI
 isPositif x = x >= 0
```

```
-- APLIKASI
-- isPositif 1
```

Apakah Huruf A → IsAnA

- Memeriksa apakah sebuah huruf (karakter) adalah huruf A atau bukan
- Misalkan karakternya adalah c, jika c adalah huruf A maka nilai fungsi IsAnA adalah benar
- Penulisan dalam Notasi Haskell

```
-- APAKAH HURUF A                    isAnA c
```

```
-- DEFINISI DAN SPESIFIKASI
isAnA :: Char -> Bool
 -- isAnA c benar jika c adalah karakter (huruf) 'A'
```

```
-- REALISASI
isAnA c  =  c == 'A' || c=='a'
```

```
-- APLIKASI
   -- isAnA 'A'
```

Apakah Origin → IsOrigin

- Memeriksa apakah dua bilangan integer (absis x dan ordinat y) adalah titik O (0,0) atau bukan
- IsOrigin adalah benar jika x adalah 0 dan y adalah 0
- Penulisan dalam Notasi Haskell

```
-- APAKAH ORIGIN                          isOrigin x y
```

```
-- DEFINISI DAN SPESIFIKASI
isOrigin :: Int -> Int -> Bool
   -- isOrigin x y benar jika x dan y adalah dua nilai yang
   -- mewakili titik origin (0,0)
```

```
-- REALISASI
isOrigin x y = (x == 0) && (y == 0)
```

```
-- APLIKASI
-- isOrigin 1 0
```

Apakah Valid → IsValid

- Memeriksa apakah sebuah bilangan integer (x) valid atau tidak. Valid jika x bernilai lebih kecil dari 5 atau lebih besar dari 500
- IsValid adalah benar jika x < 5 atau x > 500
- Penulisan dalam Notasi Haskell

```
-- APAKAH VALID                                        isValid x
```

```
-- DEFINISI DAN SPESIFIKASI
isValid :: Int -> Bool
  -- isValid x benar jika x bernilai lebih kecil dari 5
  -- atau lebih besar dari 500
```

```
-- REALISASI
isValid x =  x < 5 || x > 500
```

```
-- APLIKASI
 -- isValid 0
```

Latihan 1

- Buatlah realisasi dari fungsi di bawah ini berdasarkan definisi dan spesifikasi yang diberikan.

```
-- APAKAH JAM VALID?                              isJamValid j m d
-- DEFINISI DAN SPESIFIKASI
isJamValid :: Int -> Int -> Int -> Bool
{- IsJamValid j m d menghasilkan nilai true jika j, m, d menyusun jam
   yang valid. Definisi jam yang valid adalah jika elemen jam (j)
   bernilai antara 0 dan 23, elemen menit (m) bernilai antara 0 dan 59,
   dan elemen detik (d) bernilai antara 0 dan 59. -}
```

Latihan 2

- Diberikan 3 buah integer j, m, d dengan j adalah integer [0..23], m adalah integer [0..59], d adalah integer [0..59], yang artinya adalah jam (j), menit (m), dan detik (d) pada suatu tanggal tertentu.
- Hitunglah jumlah detik dari jam tersebut terhitung mulai jam 0:0:0 pada tanggal ybs.

Ekspresi Bernama dan Nama Antara

- Ekspresi "antara" → nama yang digunakan sementara dalam fungsi, hanya berlaku dalam fungsi, tidak di dunia luar

```
<NAMA-FUNGSI> <list-parameter> =
    let   <Nama-1> = <Ekspresi-1>
          <Nama-2> = <Ekspresi-2>
          ...
          <Nama-k> = <Ekspresi-k>
    in
          <Ekspresi-fungsional yang memanfaatkan
            Nama-1, Nama-2,..., Nama-k>
```

Kasus-1: Mean Olympique

- Definisikan sebuah fungsi yang menerima 4 bilangan positif, menghasilkan harga rata-rata dari dua di antara empat buah bilangan tersebut, dengan mengabaikan nilai terbesar dan nilai terkecil.
- Ide (dengan kalkulasi): MO = (jumlah ke empat angka, dikurangi dengan terbesar, dikurangi dengan terkecil) dibagi dua.

```
-- Mean Olympique                                   mo a b c d
```

```
-- DEFINISI DAN SPESIFIKASI
mo :: Float -> Float -> Float -> Float -> Float
{-  mo a b c d menghasilkan harga rata-rata dari dua di
    antara a, b, c, d, dengan mengabaikan nilai terbesar dan
    nilai  terkecil  -}
```

```
-- REALISASI
-- versi tanpa “abstraksi”
mo a b c d =
      let maxab = (a+b + abs(a-b))/2
          maxcd = (c+d + abs(c-d))/2
          minab = (a+b - abs(a-b))/2
          mincd = (c+d - abs(c-d))/2
      in
            let maks = (maxab+maxcd + abs(maxab-maxcd))/2
                min = (minab+mincd - abs(minab-mincd))/2
            in
                  (a+b+c+d-maks-min)/2
```

```
-- APLIKASI
-- mo 7 9 6 9
```

```
-- Mean Olympique                                   mo2 a b c d
-- DEFINISI DAN SPESIFIKASI
mo2 :: Float -> Float -> Float -> Float -> Float
-- mo2 a b c d menghasilkan harga rata-rata dari dua di
-- antara a, b, c, d, dengan mengabaikan nilai terbesar dan
-- nilai  terkecil
max4 :: Float -> Float -> Float -> Float  -> Float
-- max4 a b c d menghasilkan maksimum dari a, b, c, d
min4 :: Float -> Float -> Float -> Float  -> Float
-- min4 a b c d menghasilkan minimum dari a, b, c, d
max2 :: Float -> Float -> Float
-- max2 a b menghasilkan maksimum dari a dan b
min2 :: Float -> Float -> Float
-- min2 a b menghasilkan minimum dari a dan b
```

Pada prakteknya, sering kali Max4 dan Min4 sudah tersedia (sehingga tinggal dipakai)

```
-- REALISASI { versi dengan "abstraksi" }
max2 a b = (a+b+abs(a-b))/2
min2 a b = (a+b-abs(a-b))/2
max4 a b c d = max2 (max2 a b) (max2 c d)
min4 a b c d = min2 (min2 a b) (min2 c d)
mo2 a b c d = (a+b+c+d - (max4 a b c d) - (min4 a b c d))/2
```

```
-- Mean Olympique                                   mo2 a b c d
-- DEFINISI DAN SPESIFIKASI
mo2 :: Int -> Int -> Int -> Int -> Float
{-  mo2 a b c d menghasilkan harga rata-rata dari dua di
    antara a, b, c, d, dengan mengabaikan nilai terbesar dan
    nilai  terkecil  -}
max4 :: Int -> Int -> Int -> Int -> Int
-- max4 a b c d menghasilkan maksimum dari a, b, c, d
min4 :: Int -> Int -> Int -> Int -> Int
-- min4 a b c d menghasilkan minimum dari a, b, c, d
max2 :: Int -> Int -> Int
-- max2 a b menghasilkan maksimum dari a, b
min2 :: Int -> Int -> Int
-- min2 a b menghasilkan minimum dari a, b
-- REALISASI  {versi dengan "abstraksi"}
max2 a b = div (a+b+abs(a-b)) 2
min2 a b = div (a+b-abs(a-b)) 2
max4 a b c d = max2 (max2 a b) (max2 c d)
min4 a b c d = min2 (min2 c d) (min2 c d)
mo2 a b c d = fromIntegral(a+b+c+d-(max4 a b c d)-(min4 a b c d))/2
```

fromIntegral untuk typecasting dari type Int menjadi type Float

Ekspresi Kondisional

- Ekspresi kondisional:
  - ekspresi yang keluarannya tergantung kepada hasil evaluasi beberapa kondisi
  - hasil analisis kasus (dekomposisi persoalan, pemecahan secara independen)
- Setiap kasus harus disjoint dan analisa kasus harus mencakup semua kasus

Ekspresi Kondisional (cont.)
Notasi dan Evaluasi

- Jenis-jenis notasi kondisional:
  - Notasi if-then-else untuk 2 kasus saling komplementer
  - Notasi depend on
  - Notasi depend on dengan else (otherwise)
- Tidak ada notasi if-then karena semua kasus harus ada nilainya → fungsi harus mengembalikan nilai, termasuk ketika kondisi dalam "if" tidak terpenuhi
- Urutan kondisi tidak penting (komutatif)
- Evaluasi ekspresi parsial (hanya untuk kondisi yang evaluasinya true)

Dua Kasus Saling Komplementer

- Persoalan mencari nilai maksimum dari dua buah nilai integer, misal: a dan b
  Kasus 1: $a \geq b$
  Kasus 2: $a < b$
- Kasus 1 dan 2 disjoint.
- Tidak ada kasus lain selain 2 kasus tsb.

Dua Kasus Saling Komplementer (cont.) if-then-else

```
<Nama-fungsi> <list parameters> =
      if  <kondisi-1>  then
          <ekspresi-1>
      else   -- not kondisi-1
          <ekspresi-2>
```

```
max2 a b = if (a>=b) then
               a
           else -- a<b
               b
```

```
-- Aplikasi
-- max2 (-1) 5
```

Dua Kasus Saling Komplementer (cont.) depend-on dengan otherwise

- Pada Haskell, notasi "depend on" menggunakan **guard**/ tanda *pipe* (|)
- Template:

```
<Nama-fungsi> <list parameters>
      |  <kondisi-1> = <ekspresi-1>
      |  otherwise   = <ekspresi-2>
```

```
max2 a b
   | (a >= b)  = a
   | otherwise = b  -- a<b
```

```
-- Aplikasi
-- max2 7 (-5)
```

Analisis Kasus: Lebih Dari 2 Kasus

- Persoalan mencari nilai maksimum dari tiga buah nilai integer yang tidak sama, misal: a, b, c
- Kasus 1 : a > b and a > c
- Kasus 2 : b > a and b > c
- Kasus 3 : c > a and c > b
- Kasus 1, 2, 3 disjoint dan lengkap

Analisis Kasus: Lebih Dari 2 Kasus (cont.) if-then-else

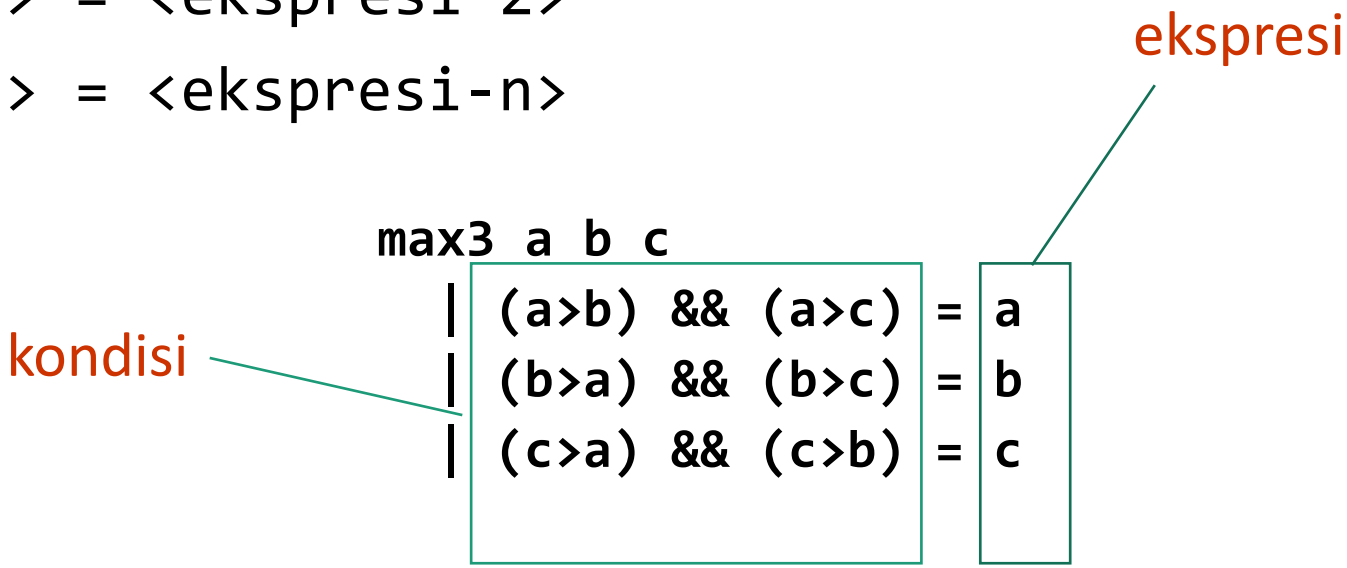
```
<Nama-fungsi> <list parameters> =
    if   <kondisi-1> then
         <ekspresi-1>
    else if <kondisi-2> then
         <ekspresi-2>
    else <ekspresi-n>
```

Tiga buah integer yang nilainya berbeda (tidak ada yang sama)

```
max3 a b c =
   if (a>b) && (a>c) then a
   else if (b>a) && (b>c) then b
   else c
```

Analisis Kasus: Lebih Dari 2 Kasus (cont.) depend-on

```
<Nama-fungsi> <list parameter>
    | <kondisi-1> = <ekspresi-1>
    | <kondisi-2> = <ekspresi-2>
    | <kondisi-n> = <ekspresi-n>
```



Kasus-2: Maksimum 3 bilangan

- Buatlah definisi, spesifikasi dan realisasi fungsi max3 yang menerima 3 buah bilangan bulat dan menghasilkan maksimum dari ke-3 bilangan tsb.
  - Versi 1: identifikasi domain, berangkat dari hasil
  - Versi 2: berdasarkan letak ketiga bilangan pada sumbu bilangan (terdapat 6 kemungkinan kasus)
  - Versi 3: reduksi domain (bandingkan 2 nilai, lalu bandingkan hasilnya dgn nilai ke-3)
  - Versi 4: memanfaatkan fungsi max2, aplikasikan max2 sebanyak 2 kali

Maksimum 3 Bilangan (versi 1)

Identifikasi domain, berangkat dari hasil.
Pilih bilangan yang maksimum dan tentukan ekspresi yang menyatakan bilangan tersebut maksimum.

```
-- MAKSIMUM 3 BILANGAN (versi 1)                    max3 a b c
```

```
-- DEFINISI DAN SPESIFIKASI
max3 :: Int -> Int -> Int -> Int
{- max3 a b c menentukan maksimum dari 3 bilangan integer yang
   berlainan nilainya, a != b dan a != c dan b != c -}
```

```
-- REALISASI
max3 a b c
   | (a>b) && (a>c) = a
   | (b>a) && (b>c) = b
   | (c>a) && (c>b) = c
```

Maksimum 3 Bilangan (versi 3)

Reduksi domain fungsi: Ambil 2 dari 3 nilai, bandingkan. Manfaatkan hasil perbandingan untuk menentukan maksimum dengan cara membandingkan terhadap bilangan ketiga.

```
-- MAKSIMUM 3 BILANGAN (versi 3)                    max3 a b c
```

```
-- DEFINISI DAN SPESIFIKASI
max3 :: Int -> Int -> Int -> Int
{- max3 a b c menentukan maksimum dari 3 bilangan integer yang
   berlainan nilainya, a != b dan a != c dan b != c -}
```

```
-- REALISASI
max3 a b c = if (a > b) then
                 if (a > c) then a
                 else c
             else -- a < b
                 if (b > c) then b
                 else c
```

AND THEN dan OR ELSE

- Operator AND THEN
  - Ekspresi A AND then B: B hanya dievaluasi jika Ekspresi A bernilai true.
- OPERATOR OR ELSE
  - B hanya dievaluasi jika Ekspresi A bernilai false.
- Dalam Haskell tidak ada notasi khusus untuk AND THEN dan OR ELSE → menggunakan bentuk ekivalennya seperti pada tabel di bawah

Ekpresi boolean	Ekivalen dengan
A **and then** B	**if** A **then** B **else** **false**
A **or else** B	**if** A **then** **true** **else** B

Latihan 3

- Buatlah realisasi dari fungsi di bawah ini berdasarkan definisi dan spesifikasi yang diberikan. Jika perlu membuat fungsi antara, buatlah definisi, spesifikasi dan realisasinya.

```
-- APAKAH DATE VALID?                          isDateValid d m y
-- DEFINISI DAN SPESIFIKASI
isDateValid ::  Int -> Int -> Int -> Bool
{- isDateValid d m y mengembalikan nilai true jika
   d, m, y membentuk date yang valid. Definisi date yang
   valid adalah jika elemen hari (d) bernilai antara 1
   dan 31, tergantung pada bulan dan apakah tahun kabisat
   atau bukan, elemen bulan (m) bernilai antara 1 dan 12,
   dan elemen tahun (y) bernilai antara 0 dan 99.
   Nilai y mewakili tahun 1900 s.d. 1999 -}
```

Latihan 4

- Buatlah definisi, spesifikasi, dan realisasi dari fungsi **nilaiTengah** yang menerima masukan 3 buah integer yang berlainan nilainya yang urutannya bisa acak dan mengembalikan sebuah integer yang merupakan salah satu dari ke-3 nilai tersebut yang jika diurutkan berada di tengah.
- Contoh aplikasi fungsi pada Haskell:

```
> nilaiTengah 1 2 3
2
> nilaiTengah (-6) 1 5
1
> nilaiTengah (-1) (-4) 10
-1
```

- Petunjuk: Buatlah fungsi min3 dan max3 dalam realisasinya.

Fungsi dapat mengembalikan tuple (pasangan nilai)

- Fungsi dapat mengembalikan range type bentukan tanpa nama
  - Dalam Haskell ditulis dalam bentuk tuple: (x1, x2, … xn)
- Contoh: Ekivalensi detik ke hari, jam, menit, detik
  - Diberikan sebuah besaran integer positif yang mewakili nilai detik.
  - Tuliskan fungsi hhmmdd yang menghasilkan nilai hari, jam, menit, detik dari besaran detik tsb
  - Contoh: Diberikan 309639, dihasilkan 3 hari 14 jam 0 menit 39 detik

```
hhmmdd x =  ( div x 86400,
              div (mod x 86400) 3600,
              div (mod (mod x 86400) 3600) 60,
              mod (mod (mod x 86400) 3600) 60  )
```

Latihan 5

- Mata uang US adalah dollar. 1 dollar = 100 sen. Dalam mata uang di US dikenal beberapa jenis koin yang masing-masing diberi nama yaitu quarter (1 quarter = 25 sen = 0,25 dollar), dime (1 dime = 10 sen = 0,1 dollar), nickel (1 nickel = 5 sen = 0,05 dollar), dan penny (1 penny = 1 sen = 0,01 dollar). Buatlah sebuah fungsi dalam Haskell yang menerima input sejumlah koin quarter, dime, nickel, dan penny dan menghasilkan berapa dollar dan berapa sen yang senilai dengan total koin-koin tersebut.
- Contoh: 8 quarter, 20 dime, 30 nickel, dan 77 penny adalah sama dengan 6 dollar dan 27 sen.
- Perhatikan bahwa output yang dihasilkan adalah pasangan nilai <dollar, sen>.

Apa yang sudah dipelajari?

- Abstraksi persoalan dengan menggunakan pendekatan berpikir fungsional
- Membuat program kecil dalam notasi fungsional dan notasi Haskell yang melibatkan ekspresi aritmatika, logika, dan kondisional

Bahan

- Diktat “Dasar Pemrograman, Bag. Pemrograman Fungsional” oleh Inggriani Liem, revisi Februari 2014
- “Haskell The Craft of Functional Programming”; Simon Thompson; Second Edition; Addison-Wesley
- “Real World Haskell”; B. O’Sullivan, J. Goerzen, D. Stewart; O’Reilly