EKSPRESI KONDISIONAL

Ekspresi kondisional adalah suatu ekspresi yang hasil evaluasinya tergantung kepada hasil evaluasi beberapa kondisi. Karena itu, dikatakan bahwa ekspresi kondisional ditulis dengan melakukan analisa kasus.

Analisa kasus adalah salah satu bentuk DEKOMPOSISI dari satu persoalan menjadi beberapa sub-persoalan, yang ingin dipecahkan secara independent (tak saling bergantung) satu sama lain.
Objektif analisa kasus adalah mendefinisikan partisi dari domain fungsi-fungsi yang akan merupakan solusi, dengan cara melakukan emunerasi dari semua kasus.
Sebuah kasus adalah restriksi batasan dari problema dalam sebuah subdomain.
Pada bahasa pemrograman, kasus seringkali disebut sebagai **KONDISI**, yang merupakan ekspresi bernilai boolean.

Menentukan kasus

Setiap kasus harus *disjoint* (terpisah satu sama lain, tidak saling beririsan) dan analisa kasus harus mencakup semua kasus yang mungkin.
kesalahan analisa kasus yang tipikal :

- ada yang tidak tertulis
- tidak *disjoint*

Analisa kasus adalah cara menyelesaikan persoalan yang umum dilakukan, dan sering dikaitkan dengan komposisi kondisional.

Notasi Ekspresi Kondisional

Analisa kasus dalam notasi fungsional dituliskan sebagai ekspresi kondisional; dengan notasi **depend on** sebagai berikut :

```
depend on {deskripsi domain}
       <Kondisi-1>   : <Ekspresi-1>
       <Kondisi-2> : <Ekspresi-2>
       <Kondisi-3> : <Ekspresi-3>
```

Kondisi adalah suatu ekspresi boolean, dan Ekspresi adalah ekspresi fungsional
Dalam suatu ekspresi kondisional, diperbolehkan untuk menuliskan suatu kondisi khusus, yaitu **else** yang artinya "negasi dari yang pernah disebutkan"

```
depend on {deskripsi domain}
       <Kondisi-1>   : <Ekspresi-1>
 <Kondisi-2> : <Ekspresi-2>
 <Kondisi-3> : <Ekspresi-3>
 else        : <Ekspresi-4>
```

artinya ekivalen dengan

```
depend on {deskripsi domain}
       <Kondisi-1>   : <Ekspresi-1>
 <Kondisi-2> : <Ekspresi-2>
 <Kondisi-3> : <Ekspresi-3>
 not <Kondisi-1> and not  <Kondisi-2> and not <Kondisi-3>:
        <Ekspresi-4>
```

Khusus untuk **dua kasus yang saling komplementer,** notasi kondisional juga dapat dituliskan dengan notasi **IF-THEN-ELSE** sebagai berikut**:**

```
if <Kondisi-1>  then
      <Ekspresi-1>
else <Ekspresi-2>
```

yang ekivalen dengan penulisan sebagai berikut :

```
depend on
      <Kondisi-1>   : <Ekspresi-1>
      not <Kondisi-1>   : <Ekspresi-2>
```

Evaluasi ekspresi kondisional

Evaluasi dari ekspresi kondisional adalah parsial (hanya yang true) dan seperti halnya urutan evaluasi aritmatika, urutan tidak penting (komutatif):

```
(Kondisi-1 or Kondisi2- or Kondisi-3) and
not (Kondisi-1 and Kondisi-2) and
not  (Kondisi-1 and Kondisi-3) and
not  (Kondisi-2 and Kondisi-3)
```

Karena harus *disjoint*, maka semua kondisi harus mutual exclusive

Operator Boolean AND then, OR else

Di samping operator boolean AND dan OR yang pernah disebutkan sebelumnya, berdasarkan urut-urutan evaluasi ekspresinya, maka beberapa bahasa menyediakan operator **AND then** dan **OR else**, yang akan diuraikan artinya pada bagian ini. Operator-operator boolean tambahan ini sengaja diuraikan pada bagian analisa kasus, karena erat kaitannya dengan ekspresi kondisional.

Operator AND then

Ekspresi A AND then B : B hanya dievaluasi jika Ekspresi A bernilai true.

Ekspresi boolean	Ekivalen dengan
A **AND then** B	**if** A **then** B **else** false

OPERATOR OR else

B hanya dievaluasi jika Ekspresi A bernilai false.

Ekspresi boolean	Ekivalen dengan
A **OR else** B	**if** A **then true else** B

Catatan :

- banyak bahasa hanya menyediakan if-then-else, maka notasi depend on harus diterjemahkan ke dalam if-then-else. Namun, untuk kejelasan teks dan kemudahan pembacaan, untuk menuliskan banyak kasus pada notasi fungsional harus dipakai **depend on**. Notasi **If-then-else** hanya dipakai untuk kasus komplementer.
- kesulitan else : kasus tidak dinyatakan secara eksplisit
- if - then -

  tidak ada artinya untuk ekpresi fungsional
  karena harus ada nilai untuk semua kasus.

Ekspresi kondisional yang menghasilkan nilai boolean sebagai berikut

depend on {deskripsi domain}
<Kondisi-1> : *<Ekspresi-1>*
<Kondisi-2> : *<Ekspresi-2>*
<Kondisi-3> : *<Ekspresi-3>*

dapat ditulis dalam bentuk lain yang ekivalen sebagai berikut.

depend on {deskripsi domain}
<Kondisi-1> **AND then** *<Ekspresi-1>* **or**
<Kondisi-2> **AND then** *<Ekspresi-2>* **or**
<Kondisi-3> **AND then** *<Ekspresi-3>*

Jika urutan penting, maka OR bisa ditulis dengan OR else dan pemakaian tanda kurung

Contoh :

depend on
<*Kondisi-1*> : true
<*Kondisi-2*> : false
<*Kondisi-3*> : <*Ekspresi-3*>

dapat ditulis

<*Kondisi-1*> **or** (<*Kondisi-2*> **AND then** <*Kondisi-3*>)

Contoh-1 Ekspresi kondisional : MAKSIMUM 2 NILAI

Persoalan :
Buatlah definisi, spesifikasi dan realisasi dari fungsi yang menghasilkan nilai maksimum dari dua buah nilai integer yang diberikan

MAKSIMUM 2 NILAI **max2(a,b)**

DEFINISI DAN SPESIFIKASI

max2 : 2 integer → integer

(max2 (a,b) menghasilkan maksimum dari 2 bilangan integer a dan b }

REALISASI

```
{ Notasi if then else sebab hanya ada dua kasus komplementer... }
      max2 (a,b) :
         if a≥b then a
               else b
```

Contoh-2 Ekspresi kondisional : MAKSIMUM 3 NILAI

Persoalan :
Buatlah definisi, spesifikasi dan realisasi dari fungsi yang menghasilkan nilai maksimum dari tiga buah nilai integer yang berlainan.

MAKSIMUM 3 NILAI **max3(a,b,c)**

DEFINISI DAN SPESIFIKASI

max3 : 3 integer → integer

(max3(a,b,c)menentukan maksimum dari 3 bilangan integer yang berlainan nilainya, $a \neq b$ dan $b \neq c$ dan $a \neq c$ }

Versi 1 : Identifikasi domain , berangkat dari hasil :

MAKSIMUM 3 NILAI (versi 1) **max3(a,b,c)**

DEFINISI DAN SPESIFIKASI

max3 : 3 integer → integer

(max3(a,b,c)menentukan maksimum dari 3 bilangan integer yang berlainan nilainya, $a \neq b$ dan $b \neq c$ dan $a \neq c$ }

REALISASI

```
max3 (a,b,c) :
   depend on a,b,c
        a>b and a>c: a
        b>a and b>c: b
        c>a and c>b: c
```

Versi2 : berdasarkan analisis letak ke tiga bilangan pada Sumbu bilangan , berangkat dari hasil. Karena data adalah 3 bilangan positif, dibagi sesuai posisi bilangan pada sumbu bilangan. Maka ada enam kemungkinan letak ke tiga nilai tsb pada sumbu bilangan.

MAKSIMUM 3 NILAI (versi 2) **max3(a,b,c)**

DEFINISI DAN SPESIFIKASI

max3 : 3 integer → integer

(max3(a,b,c)menentukan maksimum dari 3 bilangan integer yang berlainan nilainya, $a \neq b$ dan $b \neq c$ dan $a \neq c$ }

REALISASI

```
max3 (a,b,c) :
      depend on a,b,c
          a>b and b>c: a
          a>c and c>b: a
          b>a and a>c: b
          b>c and c>a: b
          c>a and a>b: c
          c>b and b>a: c
```

Versi 3 : Reduksi dari domain fungsi- fungsi

- ambil dua dari tiga nilai yang akan dicari maksimumnya, bandingkan
- manfaatkan hasil perbandingan untuk menentukan maksimum dengan cara membandingkan terhadap bilangan ke tiga.

MAKSIMUM 3 NILAI (versi 3) **max3(a,b,c)**

DEFINISI DAN SPESIFIKASI

max3 : 3 integer → integer

(max3(a,b,c)menentukan maksimum dari 3 bilangan integer yang berlainan nilainya, $a \neq b$ dan $b \neq c$ dan $a \neq c$ }

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
```

Versi 4 : reduksi domain seperti pada versi 3 tetapi dengan nama antara

MAKSIMUM 3 NILAI (versi 4) **max3(a,b,c)**

DEFINISI DAN SPESIFIKASI

max3 : tiga integer → integer

(max3(a,b,c)menentukan maksimum dari 3 bilangan integer yang berlainan nilainya, $a \neq b$ dan $b \neq c$ dan $a \neq c$ }

REALISASI

```
max3(a,b,c)
     let m = depend on a,b
                       a>b:a
                       a<b:b
         in
       depend on m,c
          m>c:m
          m<c:c
```

Versi-5 : aplikasi suatu fungsi yang pernah dibuat.
Dengan sudah tersedianya MAX2, maka MAX3 dapat dituliskan dengan aplikasi dari MAX2, salah satu cara apalikasi adalah sebagai berikut :

MAKSIMUM 3 NILAI (versi 5) **max3 (a,b,c)**

DEFINISI DAN SPESIFIKASI

max3 : 3 integer → integer

(max3(a,b,c)menentukan maksimum dari 3 bilangan integer yang berlainan nilainya, $a \neq b$ dan $b \neq c$ dan $a \neq c$ }

REALISASI

```
max3(a,b,c) : max2(max2(a,b,),c)
```

Versi 6 : idem dengan versi 5, dengan cara aplikasi yang berbeda

MAKSIMUM 3 NILAI (versi 6)	**max3 (a,b,c)**

DEFINISI DAN SPESIFIKASI

max3 : 3 integer → integer

(max3(a,b,c)menentukan maksimum dari 3 bilangan integer yang berlainan nilainya, $a \neq b$ dan $b \neq c$ dan $a \neq c$ }

REALISASI

```
max3(a,b,c) : max2(c, max2(a,b))
```

MAKSIMUM 3 NILAI (versi 7)	**max3 (a,b,c)**

DEFINISI DAN SPESIFIKASI

max3 : 3 integer → integer

(max3(a,b,c)menentukan maksimum dari 3 bilangan integer yang berlainan nilainya, $a \neq b$ dan $b \neq c$ dan $a \neq c$ }

REALISASI

```
max3(a,b,c) : max2(b, max2(a,c))
```

Contoh3 Ekspresi kondisional : PENANGGALAN

Persoalan :

Tanggal, bulan dan tahun pada perioda tahun 1900 s/d 1999 dapat dituliskan dalam "tuple" dari tiga buah bilangan integer <d,m,y> sebagai berikut :

<3,4,93> : hari ke 3, pada bulan ke 4 (April), pada tahun 1993

Hitunglah hari ke... pada **suatu tahun 1900+y** mula-mula tanpa memperhitungkan adanya tahun kabisat, kemudian dengan memperhitungkan tahun kabisat.

Contoh : <1,1,82> → 1
<31,12,72> → 366
<3,4,93> → 93

Versi 1 : Tanpa memperhitungkan kabisat

PENANGGALAN	**HariKe1900(d,m,y)**

DEFINISI DAN SPESIFIKASI

Harike1900 : integer [1..31], integer [1..12] integer [0..99] → integer [1..366]

{Harike1900(d,m,y) dari suatu tanggal <d,m,y> adalah hari 'absolut' dihitung mulai 1 Januari 1900+y. 1 Januari tahun 1900+y adalah hari ke 1}

dpm : integer [1..12] → integer [1..36]

{dpm(B) adalah jumlah hari pada tahun ybs pada tanggal 1 bulan B. terhitung mulai satu januari: kumulatif jumlah hari dari tanggal 1 Januari s/d tanggal 1 bulan B, tanpa memperhhitungkan tahun kabisat}

REALISASI { TANPA KABISAT }

```
Harike1900 (d,m,y) :
   dpm (m) + d - 1
dpm (B) : { analisa kasus terhadap B }
   depend on B
         B = 1: 1
         B = 2: 32
         B = 3: 60
         B = 4: 91
         B = 5: 121
         B = 6: 152
         B = 7: 182
         B = 8: 213
         B = 9: 244
         B = 10: 274
         B = 11: 305
         B = 12: 335
```

Versi-2 : dengan memperhitungkan tahun kabisat

PENANGGALAN **HariKe1900(d,m,y)**

DEFINISI DAN SPESIFIKASI

Harike1900 : integer [1..31], integer [1..12], integer [0..99] → integer [1..366]

{Harike1900 (d,m,y) dari suatu tanggal <d,m,y> adalah hari 'absolut' dihitung mulai 1 Januari tahun ke 1900+ y. 1 Januari tahun 1900+y adalah hari ke 1 }

dpm : integer [1..12] → integer [1..36]

{dpm(B) adalah jumlah hari pada tahun ybs pada tanggal 1 bulan B. terhitung mulai satu januari: kumulatif jumlah hari dari tanggal 1 Januari s/d tanggal 1 bulan B, tanpa memperhhitungkan tahun kabisat}

IsKabisat? : integer [0..99] → boolean

{IsKabisat?(a) true jika tahun 1900+a adalah tahun kabisat: habis dibagi 4 tetapi tidak habis dibagi 100, atau habis dibagi 400 }

REALISASI {dengan memperhitungkan tahun kabisat}

```
{ Realisasi dpm(B) sama dengan pada versi ke 1 }
IsKabisat?(a):( (a mod 4 = 0) and (a mod 100 ≠ 0 ) ) or (a div 400 =0)

Harike1900 (d,m,y) :
      dpm (m) + d - 1 +
          (if m > 2 and IsKabisat? (y)
               then 1 else 0)
```

Atau dapat ditulis :

REALISASI

```
Harike1900(d,m,y) :
  dpm (m) + d - 1 +
     if m>2 AND then
          kabisat (y) then  1
     else 0
```

Latihan soal :

1. Bagaimana jika nilai yang harus dicari maksimumnya bukan hanya 3 tetapi dibuat umum sehingga N, dengan N adalah bilangan positif ? Dengan ide Solusi MAX3, yang mana paling gampang diadaptasi/ dipakai ? Solusi versi 2 adalah dasar dari definisi rekurens : max dari n bilangan adalah maximum dari 2 bilangan, salah satunya adalah maximum dari n-1 yang lain
2. Modifikasilah HariKe1900(d,my) sehingga menghasilkan hari absolut terhitung mulai 1 Januari 1900. Jadi 1 Januari 1900 adalah hari ke 1.
3. Diberikan sebuah tuple <j,m,s> dengan j bilangan integr [0..24], m bilangan integer[0..59] dan s bilangan integer[0..59] yang artinya adalah jam, menit dan detik pada suatu tanggal tertentu.
4. Hitunglah detik dari jam tersebut terhitung mulai jam 00:00:00 tanggal yang bersangkutan.
5. Cetaklah seperti penulisan jam digital yang mampu untuk menulis dengan jam di antara 0..12 saja namun dituliskan dengan 'pm' atau 'am' sebagai berikut : ditulis sebagai <2,20,15> pm
6. Tuliskanlah sebuah fungsi yang menerima suatu besaran dalam derajat Celcius dan kode konversi ke derajat Reamur, Fahrenheit atau Kelvin, dan mengirimkan nilai derajat sesuai dengan kode konversi.
7. Diberikan suatu besaran yang menyatakan temperatur air dalam derajat Celcius dan pada tekanan 1 atm. Harus ditentukan apakah air tersebut berwujud es (padat), cair atau uap.
8. Tuliskanlah sebuah fungsi yang harus mengirimkan kode diet pasien, yang menerima masukan berat badan dan jumlah kalori yang dibutuhkan pasien tersebut. Spesifikasikan dengan jelas hubungan antara kode jenis diet dengan berat badan dan jumlah kalori!

TYPE BENTUKAN
(Produk, Type Komposisi, Type Terstruktur)

Pada pembahasan sebelumnya, semua program fungsional mempunyai ***domain*** dan ***range*** type dasar (numerik, karakter, boolean). Pada bagian ini akan dibahas mengenai produk ***(product)*** dari tipe, yang dalam beberapa paradigma dan bahasa pemrograman biasa disebut sebagai **type bentukan, type komposisi, type terstruktur,** ***record***. Untuk selanjutnya, dalam diktat ini disebut sebagai type bentukan.
Pokok bahasan dari bab ini adalah :

1. Bagaimana memakai ekspresi fungsional untuk mendefinisikan type komposisi/ terstruktur
2. Product dari type
3. Konstruktor dan selektor
4. Predikat dan fungsi lain terhadap type

Definisi type :

Type adalah himpunan nilai dan sekumpulan operator terdefinisi terhadap type tersebut. Membuat definisi dan spesifikasi type adalah menentukan nama, domain dan operasi yang dapat dilakukan terhadap type tersebut. Dalam konteks fungsional, operator terhadap type dijabarkan menjadi fungsi. Realisasi dan aplikasi terhadap fungsi yang mewakili "type" ditentukan oleh bahasa pemrograman nyatanya. Pada diktat ini, hanya diberikan definisi dan spesifikasi.

Nilai type bentukan
Domain nilai suatu nama bertype bentukan oleh domain nilai komponennya.
Karena merupakan product, maka nilai suatu type bentukan dituliskan dalam tuple, sesuai dengan komponen pembentuknya. Contoh : Untuk menyatakan **nilai** suatu Point yang didefinisikan oleh ***tuple*** <x:integer, y:integer>, dipakai notasi : <0,0> sebagai Titik Origin, <1,2> untuk Titik dengan x=2 dan y=2

Contoh :

integer : deret dari bit
word : deret dari character , deret caracter 8 bit
date : integer yang mewakili hari dibanding suatu reference.

Type bentukan dapat dibentuk dari type yang tersedia (type dasar), atau dari type yang pernah didefinisikan.

Beberapa contoh type bentukan yang sering dipakai dalam pemrograman:

1. Type Point, yang terdiri dari <absis,ordinat> bertype <integer,integer>
2. Type Bilangan Kompleks, yang terdiri dari bagian riil dan bagian imaginer: yang bertype bilangan riil<riil,imaginer>
3. Type Pecahan, yang terdiri dari <pembilang, penyebut> yang bernilai <integer, integer>
4. Type JAM, yang terdiri dari< jam,menit,detik>
5. Type DATE yang terdiri dari <tangal,bulan,tahun>

Dari suatu Type bentukan, dapat dibentuk type bentukan yang lain. Maka dalam hal ini, komponen type bentukan adalah type bentukan. Misalnya :

1. Berdasarkan type Point <absis, ordinat>, dapat dibentuk type berikut:
   a) Garis, yang terdiri dari <titik-awal, titik-akhir>
   b) Segiempat yang terdiri dari <Titik-Top, Titik-Bottom>
2. Berdasarkan JAM dan DATE, dapat dibentuk type baru WAKTU yang terdiri dari <jam,tanggal>

Seperti telah disebutkan pada bab sebelumnya, Operator terdefinisi pada konteks fungsional adalah operator dasar aritmatika, relasional dan boolean. Dalam beberapa persoalan semacam yang berikut ini kita dihadapkan kepada persoalan yang mengharuskan pengelolaan type bentukan (type terstruktur, komposisi) yang operatornya harus didefinisikan berdasarkan operator dasar tersebut.

Pendefinisian type komposisi dalam konteks fungsional adalah mendefinisikan

- **Nama** type dan komposisi/strukturnya, hanya akan menjadi definisi
- **Selektor**, untuk mengakses komponen type komposisi menjadi elemen dasar sehingga dapat dioperasikan. Selektor ditulis definisi dan spesifikasinya sebagai **fungsi selektor.** Selektor suatu type bentukan **dalam notasi fungsional tidak direalisasi**, karena realisasinya sangat tergantung kepada ksetersediaan bahasa. Akan direalisasi langsung menjadi ekspresi dalam bahasa tertentu, misalnya dengan LISP pada bagian kedua
- **Konstruktor** untuk "membentuk" type komposisi, juga dituliskan definisi dan spesifikasinya sebagai sebuah fungsi. Nama Konstruktor biasanya diawali dengan "Make". Seperti halnya selektor, konstruktor suatu type **bentukan dalam notasi fungsional tidak direalisasi**, karena realisasinya sangat tergantung kepada ketersediaan bahasa. Akan direalisasi langsung menjadi ekspresi dalam bahasa tertentu, misalnya dengan LISP pada bagian kedua
- **Predikat** yang perlu, untuk menentukan karakteristik dan pemeriksaan besaran,
- **Fungsi-fungsi** lain yang **didefinisikan**, dibuat **spesifikasinya** dan harus **direalisasi** untuk type tersebut, yang akan berlaku sebagai "operator" terhadap type tersebut

Karena dalam konteks fungsional hanya ada fungsi, maka perhatikanlah bahwa mengolah suatu type bentukan di dalam konteks fungsional: semua objek adalah fungsi dan pada akhirnya, ketika realisasi, pengertian "type" lenyap sehingga kita tidak perlu lagi untuk merealisasikan type berkat adanya konstruktor type tsb. Ini sesuai dengan konteks fungsional, di mana semua objek yang dikelola adalah fungsi.
Realisasi fungsi tidak dilakukan untuk pendefinisian type, konstruktor dan selektor. Realisasi fungsi hanya dilakukan untuk predikat dan fungsi lain terhadap pecahan.

Pembahasan bab ini mencakup dua hal :

- Type bentukan sebagai parameter fungsi. Pembahasan akan dilakukan mula-mula melalui studi kasus untuk kebutuhan suatu fungsi sederhana, dan kemudian pembentukan modul fungsional untuk : type pecahan, type date, type Point
- Type bentukan yang merupakan hasil evaluasi suatu fungsi. Pembahasan juga akan dilakukan melalui contoh.
- Type bentukan tanpa nama, yang hanya diwakili oleh ***tuple*** dari nilai

Contoh kasus sederhana pemrosesan type terstruktur

`Kasus 1 :` Type POINT

Didefinisikan suatu type bernama Point, yang mewakili suatu titik dalam koordinat kartesian, terdiri dari absis dan ordinat.
Berikut ini adalah teks dalam notasi fungsional untuk type Point tersebut, dengan selektor yang hanya dituliskan dalam bentuk fungsi.

TYPE POINT

DEFINISI TYPE

type point : <x: real , y: real >

{<x,y> adalah sebuah point, dengan x adalah absis, y adalah ordinat }

DEFINISI DAN SPESIFIKASI SELEKTOR

Absis : point → real

{Absis(P) Memberikan Absis Point P}

Ordinat : point → real

{Ordinat(P) Memberikan ordinat Point P }

DEFINISI DAN SPESIFIKASI KONSTRUKTOR

MakePoint : 2 real → point

{ MakePoint(a,b) membentuk sebuah point dari a dan b dengan a sebagai absis dan b sebagai ordinat}

DEFINISI DAN SPESIFIKASI PREDIKAT

IsOrigin? : point → boolean

{ IsOrigin?(P) benar jika P adalah titik origin yaitu titik <0,0>}

DEFINISI OPERATOR/FUNGSI LAIN TERHADAP POINT

Jarak : 2 point → real

{Jarak(P1,P2) : menghitung jarak antara 2 point P1 dan P2 }

Jarak0 : point → integer

{ Jarak0(P1) Menghitung jarak titik terhadap titik pusat koordinat (0,0) }

Kuadran : point → integer [1..4]

{Kuadran(P) : menghitungdi mana kuadran di mana titik tersebut terletak Syarat:P bukan titik origin dan bukan terletak pada Sumbu X dan bukan terletak pada sumbu Y}

{ Fungsi antara yang dipakai : FX2 adalah pangkat dua yang pernah didefinisikan pada least square dan SQRT(X) adalah fungsi dasar untuk menghitung akar}

REALISASI

```
IsOrigin (P) : Absis(P)=0 and  Ordinat(P)=0


Jarak (P1,P2) :
      SQRT (FX2 (Absis(P1) - Absis(P2)) +
            FX2 (Ordinat(P1) - Ordinat (P2)))
 Jarak0 (P) : SQRT (FX2 (Absis(P) - Ordinat(P) ) )


 Kuadran (P) :
       depend on Absis(P), Ordinat(P):
          Absis(P) >0 and Ordinat(P) >0: 1
          Absis(P) <0 and Ordinat(P) >0: 2
          Absis(P) <0 and Ordinat(P) <0: 3
          Absis(P) >0 and Ordinat(P) <0: 4
```

Kasus-2: Type PECAHAN

Didefinisikan suatu type bernama Pecahan, yang terdiri dari pembilang dan penyebut. Berikut ini adalah teks dalam notasi fungsional untuk type pecahan tersebut.
Perhatikanlah bahwa realisasi fungsi hanya dilakukan untuk operator aritmatika dan relasional terhadap pecahan. Realisasi selektor hanya diberikan secara konseptual, karena nantinya akan diserahkan implementasinya ke bahasa pemrograman

TYPE PECAHAN

DEFINISI DAN SPESIFIKASI TYPE

type pecahan : <n: integer >=0 ,d: integer >0>

{<n:integer >=0, d:integer >0> n adalah pembilang (numerator) dan d adalah penyebut (denumerator). Penyebut sebuah pecahan tidak boleh nol }

DEFINISI DAN SPESIFIKASI SELEKTOR DENGAN FUNGSI

Pemb : pecahan → integer >=0

*{ Pemb(p) memberikan numerator pembilang **n** dari pecahan tsb }*

Peny : pecahan → integer > 0

*{ Peny(p) memberikan denumerator penyebut **d** dari pecahan tsb }*

DEFINISI DAN SPESIFIKASI KONSTRUKTOR

MakeP : integer >=0, integer > 0 → pecahan

{ MakeP(x,y) membentuk sebuah pecahan dari pembilang x dan penyebut y, dengan x dan y integer}

DEFINISI DAN SPESIFIKASI OPERATOR TERHADAP PECAHAN

{ Operator aritmatika Pecahan }

AddP : 2 pecahan → pecahan

{ AddP(P1,P2) : Menambahkan dua buah pecahan P1 dan P2 :
*n1/d1 + n2/d2 = (n1*d2 + n2*d1)/d1*d2 }*

SubP : 2 pecahan → pecahan

{ SubP(P1,P2) : Mengurangkan dua buah pecahan P1 dan P2
*Mengurangkan dua pecahan : n1/d1 - n2/d2 = (n1*d2 - n2*d1)/d1*d2 }*

MulP : 2 pecahan → pecahan

{MulP(P1,P2) : Mengalikan dua buah pecahan P1 dan P2
*Mengalikan dua pecahan : n1/d1 * n2/d2 = n1*n2/d1*d2 }*

DivP : 2 pecahan → pecahan

{ DivP(P1,P2) : Membagi dua buah pecahan P1 dan P2
*Membagi dua pecahan : (n1/d1)/(n2/d2) = n1*d2/d1*n2 }*

RealP : pecahan → real

{Menuliskan bilangan pecahan dalam notasi desimal }

DEFINISI DAN SPESIFIKASI PREDIKAT

{ Operator relasional Pecahan }

IsEqP?: 2 pecahan → boolean

{IsEqP?(p1,p2) true jika p1 = p2
Membandingkan apakah dua buah pecahan samanilainya: n1/d1 = n2/d2 jika dan hanya jika n1d2=n2d1 }

IsLtP?: 2 pecahan → boolean

{IsLtP?(p1,p2) true jika p1 < p2
Membandingkan dua buah pecahan, apakah p1 lebih kecil nilainya dari p2: n1/d1 < n2/d2 jika dan hanya jika n1d2 < n2d1 }

IsGtP?: 2 pecahan → boolean

{IsGtP?(p1,p2) tue jika p1 > p2
Membandingkan nilai dua buah pecahan,, apakah p1 lebih besar nilainya dari p2: n1/d1 > n2/d2 jika dan hanya jika n1d2 > n2d1 }

REALISASI

```
AddP(P1,P2) :
  MakeP((Pemb(P1)*Peny(P2) + Pemb(P2)*Peny(P1)),
        (Peny(P1)*Peny(P2)))
SubP(P1,P2) :
  MakeP((Pemb(P1)*Peny(P2) - Pemb(P2)*Peny(P1)),
        (Peny(P1)*Peny(P2)))
MulP(P1,P2) :
  MakeP((Pemb(P1)*Pemb(P2)),
        (Peny(P1)*Peny(P2)))
DivP(P1,P2) :
  MakeP((Pemb(P1)*Peny(P2)),
        (Peny(P1)*Pemb(P2)))
RealP(P) :
  Pemb(P)/Peny(P)
IsEqP?(P1,P2) :
    Pemb(P1)*Peny(P2)= Peny(P1)*Pemb(P2)
IsLtP?(P1,P2) :
    Pemb(P1)*Peny(P2) < Peny(P1)*Pemb(P2)
IsGtP?(P1,P2) :
    Pemb(P1)*Peny(P2) > Peny(P1)*Pemb(P2)
```

KASUS-3 : PENANGGALAN

Didefinisikan suatu type Date yang terdiri dari thari, bulan dan tahun dan membentuk komposisi <Hr,Bln,Thn >. Dalam contoh ini, sebuah nama type bukan merupakan nama type bentukan, melainkan sebuah subdomain (sebagian dari nilai domain). Penamaan semacam ini akan mempermudah pembacaan teks

TYPE DATE

DEFINISI DAN SPESIFIKASI TYPE

type Hr : integer [1...31]

{definisi ini hanyalah untuk "menamakan" type integer dengan nilai tertentu supaya mewakili hari, sehingga jika dipunyai suatu nilai integer, kita dapat memeriksa apakah nilai integer tersebut mewakili Hari yang absah }

type Bln : integer [1...12]

{definisi ini hanyalah untuk "menamakan" type integer dengan daerah nilai tertentu supaya mewakili Bulan }

type Thn : integer > 0

{definisi ini hanyalah untuk "menamakan" type integer dengan daerah nilai tertentu supaya mewakili tahun }

type date <d: Hr, m:Bln,y:Thn>

{ <d,m,y> adalah tanggal d bulan m tahun y }

DEFINISI DAN SPESIFIKASI SELEKTOR

Day : date → Hr

{Day (D) memberikan hari d dari D yang terdiri dari <d,m,y> }

Month : date → Bln

{Month (D) memberikan bulan m dari D yang terdiri dari <d,m,y> }

Year : date → Thn

{Year (D) memberikan tahun y dari D yang terdiri dari <d,m,y> }

KONSTRUKTOR

MakeDate : <Hr,Bln,Thn> → date

{MakeDate <(h,b,t>) → tgl pada hari,bulan,tahun yang bersangkutan }

DEFINISI DAN SPESIFIKASI OPERATOR/FUNGSI LAIN TERHADAP DATE

Nextday : date → date

{NextDay(D): menghitung date yang merupakan keesokan hari dari date D yang diberikan:

Nextday (<1,1,80>) adalah (<2,1,80>)
Nextday (<31,1,80>) adalah (<1,2,80>
Nextday (<30,4,80>) adalah (<1,5,80>
Nextday (<31,12,80>) adalah (<1,1,81>
Nextday (<28,2,80>) adalah (<29,2,80>
Nextday (<28,2,81>) adalah (<1,3,82> }

TYPE DATE (lanjutan)

Yesterday : date → date

{ Yesterday(D): Menghitung date yang merupakan 1 hari sebelum date D yang diberikan:

Yesterday (<1,1,80>) adalah <31,12,79>
Yesterday (<31,1,80>) adalah <30,1,80>
Yesterday (<1,5,80>) adalah <30,4,80>
Yesterday (<31,12,80>) adalah <30,12,80>
Yesterday (<29,2,80>) adalah <28,2,80>
Yesterday (<1,3,80>) adalah <29,2,80> }

NextNday : date,integer → date

{ NextNDay(D,N) : Menghitung date yang merupakan N hari (N adalah nilai integer) sesudah dari date D yang diberikan}

HariKe1900: date → integer [0..366]

{HariKe1900(D) : Menghitung jumlah hari terhadap 1 Januari pada tahun y, dengan memperhitungkan apakah y adalah tahun kabisat }

PREDIKAT

IsEqD?: 2 date → boolean

{IsEqD?(d1,d2) benar jika d1=d2, mengirimkan true jika d1=d2 and m1=m2 and y1=y2. Contoh : EqD(<1,1,90>,<1,1,90>) adalah true
EqD(<1,2,90>,<1,1,90>) adalah false }

IsBefore? : 2 date → boolean

{IsBefore?(d1,d2) benar e jika d1 adalah sebelum d2 mengirimkan true jika D1 adalah "sebelum" D2:
HariKe1900 dari D1 adalah lebih kecil dari HariKe1900 D2 }
{Contoh : Before (<1,1,80>,<1,2,80>) adalah false }
Before (<1,1,80>,<2,1,80>) adalah true }

IsAfter? : 2 date → boolean

{IsAfter?(d1,d2) benar jika d1 adalah sesudah d2 mengirimkan true jika D1 adalah "sesudah" D2: HariKe1900 dari D1 adalah lebih besar dari HariKe1900 dari D2.
Contoh : After (<1,11,80>,<1,2,80>) adalah true
After (<1,1,80>,<2,1,80>) adalah false
After (<1,1,80>,<1,1,80>) adalah false}

IsKabisat? : integer → boolean

{IsKabisat?(a) true jika tahun 1900+a adalah tahun kabisat: habis dibagi 4 tetapi tidak habis dibagi 100, atau habis dibagi 400 }

REALISASI BEFORE DENGAN FUNGSI SELEKTOR

```
Is Before? (D1,D2) :
     Let J1=Day(D1), M1=Month(D1) T1=Year(D1)
         J2=Day(D2), M2=Month(D2) T2=Year(D2) in
         if T1 ≠ T2 then T1 < T2

         else  if M1 ≠ M2  then M1<M2

               else  J1<J2
```

REALISASI BEFORE YANG LAIN

```
IsBefore? (D1,D2) :

  if Year(D1) ≠ Year(D2)
       then Year(D1) < Year(D2)
  else { tahunnya sama, bandingkan bulan }
        if Month(D1) ≠   Month(D2)
              then Month(D1) < Month(D1)
         else Day(D1) < Day(D2)
```

REALISASI

{ hanya sebagian, diberikan sebagian sebagai contoh , sisanya buatlah sebagai latihan!)

```
 Nextday (D) :
   depend on (Month(D)
      Month(D) = 1 or Month(D) = 3 or Month(D) = 5 or Month(D) = 7
      or Month(D) = 8 or Month(D) = 10: {Bulan dengan 31 hari }
           if Day(D)<31 then MakeDate(Day(D)+1,Month(D),Year(D))
                   else MakeDate(1,Month(D)+1,Year(D))
      Month(D) = 2 : {Februari}
            if Day(D)<28 then MakeDate(Day(D)+1,Month(D),Year(D))
                   else MakeDate(Day(D),Month(D)+1,Year(D))
            else if Iskabisat?(Year(D)) then
                 if Day(D)=28 then MakeDate(Day(D)+1,Month(D),Year(D))
                   else MakeDate(1,Month(D)+1,Year(D))
                 else MakeDate(1,Month(D)+1,Year(D))
      Month(D) = 4 or Month(D)=6 or Month(D)=9 or Month(D) = 11: {30 hr}
           if m<30 then MakeDate(Day(D)+1,Month(D),Year(D))
                   else MakeDate(Day(D),Month(D)+1,Year(D))
      Month(D) = 12 {Desember}
           if m<31 then MakeDate(Day(D)+1,Month(D),Year(D))
                   else MakeDate(1,1,Year(D)+1)
```

TYPE DATE (lanjutan)

REALISASI

```
Yesterday (D):
   if Day(D)=1
     depend on (Month(D)
        Month(D) = 12 or Month(D) = 3 or Month(D)=5 or Month(D) = 7
        or Month(D) = 8 or Month(D) = 10:
            MakeDate(30,Month(D) - 1,Year(D))
        Month(D) = 2 :
            if IsKabisat? (Year(D))
            then MakeDate(29,2,Year(D))
            else MakeDate(28,2,Year(D))
         Month(D) = 4 or Month(D) = 6 or  Month(D) = 9 or Month(D) = 11:
            MakeDate(31,Month(D),Year(D))
        Month(D) = 1 : MakeDate(31,12,Year(D) - 1)
    else
        MakeDate(Day(D) - 1,Month(D),Year(D))


IsEqD? (D1,D2): HariKe1900(D1) = HariKe1900(D2)
IsBefore?(D1,D2): HariKe1900(D1) < HariKe1900(D2)
IsAfter? (D1,D2): HariKe1900(D1) > HariKe1900(D2)
IsKabisat?(a):( (a mod 4 = 0) and (a mod 100 ≠ 0 ) ) or (a div 400 =0)
```

Fungsi dengan *range* type bentukan tanpa nama

Pada contoh yang diberikan di atas, semua type bentukan diberi nama. Pemberian nama type akan berguna jika type tersebut dipakai berkali-kali dan memang membutuhkan operator untuk mengoperasikan nilai-nilainya.
Bahkan seringkali, diperlukan fungsi yang menghasilkan suatu nilai bertype komposisi, tanpa perlu mendefinisikan nama tsb (jika kita mendefinisikan nama, maka kita wajib membuat konstruktor, selektor, dsb yang pernah dijelaskan).
Nama type tidak perlu didefinisikan misalnya karena kita harus merancang suatu fungsi yang hanya di-aplikasi sekali untuk menghasilkan beberapa nilai yang komposisinya mengandung arti.

Berikut ini adalah contoh yang dimaksudkan.

Kasus : **Ekivalensi detik de jam, menit, detik**
Diberikan sebuah besaran integer positif yang mewakili nilai detik, tuliskanlah sebuah fungsi HHMMDD yang memberikan nilai hari, jam, m, detik dari besaran detik tersebut.
Contoh: Diberikan 309639, menghasilkan <3,14,0,39>

Range dari HHMMDD tipe bentukan yang kebetulan terdiri 4 buah integer. Pada kasus lain, mungkin sebenarnya dapat juga berbeda-beda, misalnya kalau terdefinisi type Hr, Bln, Th seperti pada contoh DATE.

Realisasi fungsi ini membutuhkan sebuah fungsi yang mampu untuk menghitung hasil dan sekaligus sisa pembagian bulat dari dua buah bilangan integer yang dinamakan QR
Range dari QR adalah sebuah pasangan nilai integer (type terkomposisi) namun tidak diberi nama

EKIVALENSIDETIK **HHMMDD(x)**

DEFINISI DAN SPESIFIKASI

HHMMDD (x) : integer [0..99999] → <integer >0,integer[0..23],2 integer [0..59]>

*{ mis. <a,b,c,d> = HHMMDD (x), x = 86400 * a + 3600 * b + 60 * c + d }*

QR : <integer > 0, integer > 0> → < integer > 0, integer > 0 >

{let <q,r> = QR <N,D>) : q adalah hasil bagi bulat
r adalah sisa pembagian bulat dari N dibagi D
*N = q * D + r and 0 < r, r < D}*

REALISASI

```
QR (<N,D>) : < N div D, N mod D>

HHMMDD (x) :

     let < H, sisaH> = QR (x, 86400) in
       let < J, sisaJ > = QR (sisaH, 3600) in
           let <M, D> = QR (sisa J, 60) in
                <H,J,M,D>
```

REALISASI –VERSI TANPA LET

```
QR (<N,D>) : < N div D, N mod D>

HHMMDD (x) :
< x div 86400, (x mod 8600)div 3600),
     ((x mod 8600)div 3600) div 60), ((x mod 8600)div 3600) mod 60) >
```

Perhatikanlah bahwa notasi <a,b> artinya sebuah ***tuple*** yang membentuk sebuah "komposisi" dari nilai a dan b. Pendefinisian nilai bentukan semacam ini (dalam beberapa bahasa pemrograman disebut sebagai agregat) ternyata tidak disediakan primitifnya dalam semua bahasa pemrograman, berarti harus ditranslasi atau selalu dilakukan dengan mendefinisikan nama type. Notasi fungsional membolehkan penulisan ini untuk kemudahan pembacaan teks program. Selanjutnya dalam implementasi harus diterjemahkan. Contoh terjemahan kan diberikan pada Bagian kedua yaitu ke dalam bahasa LISP.

Selain type tanpa nama, beberapa masalah yang timbul berhubungan dengan *range* suatu fungsi yang merupakan komposisi nilai tapi tidak mempunyai **nama** type bentukan :

- Jika bahasa pemrograman tidak menyediakan agregat, maka kita harus merealisasi konstruktor. Ini berarti bahwa kita harus membuat nama type
- Jika bahasa pemrograman tidak menyediakan fasilitas untuk membuat fungsi dengan *range* berupa type bentukan, maka implementasi dari fungsi harus ditransalasikan melalui fasilitas yang tersedia pada bahasanya

Sebenarnya jika nilai bentukan mempunyai arti dan operator, lebih baik didefinisikan suatu nama type.

Latihan soal :

1. Definisi pecahan tsb. tidak mengharuskan bahwa pembilang lebih kecil dari penyebut. Modifikasi type pecahan tsb sehingga mampu menangani bilangan pecahan yang terdiri dari :
   <bil : integer, n: integer, d : integer >
   dengan bil : bil integer, mungkin bernilai 0 atau negatif
   n : pembilang
   d : penyebut
   dan n<d
   dan nilai pecahan adalah (bil*d + n)/d

2. Realisasikan predikat EqD, Before dan After tidak dengan melalui aplikasi Harike1900, melainkan dengan melakukan analisa kasus terhadap <d,m,y>

3. Buat type baru garis berdasarkan 2 buah point

4. Tentukan pula operator/predikat terhadap garis : misalnya apakah dua garis sejajar, menghitung sudut dari sebuah garis dihitung dari sumbu X+,

5. Buat type baru segiempat berdasarkan empat buah titik

6. Tentukan pula operator/predikat terhadap segi empat :misalnya apakah suatu titik terletak di dalam segiempat, apakah empat titik membentuk bujur sangkar, apakah empat titik membentuk sebuah jajaran genjang,

7. Buat type baru lingkaran berdasarkan sebuah titik dan sebuah garis.

8. Tentukan pula operator/predikat terhadap segi empat :misalnya apakah sebuah titik terletak didalam lingkaran, apakah lingkaran yang dibentuk berpusat pada (0,0), apakah sebuah garis memotong lingkaran, menghitung garistengah sebuah lingkaran, menghitung luas lingkaran,....

KOLEKSI OBJEK

Sampai saat ini, program fungsional hanya mampu mengelola type dasar dan type bentukan/type komposisi yang dibentuk dari type dasar. Padahal, dalam pemrograman, seringkali kita harus mengelola kumpulan objek. Perhatikanlah bahwa pada type komposisi, sebuah objek bertype komposisi (misalnya sebuah date yang merepresentasi tanggal), hanya mampu menyimpan sebuah tanggal yang terdiri dari tiga nilai integer. Kita membutuhkan suatu sarana untuk mengelola **sekumpulan** tanggal (misalnya tanggal-tanggal yang mewakili kalender dalam satu tahun).
Jika banyaknya nilai yang harus dikelola sedikit dan sudah diketahui sebelumnya, misalnya dalam kasus MAX3 (menentukan nilai maksimum dari 3 bilangan integer), tidak timbul masalah karena dalam program akan didefinisikan nama sebanyak nilai yang akan diproses. Tapi, jika dalam hal nilai yang dikelola:

- sangat banyak,
- tidak tertentu atau bervariasi, bisa banyak atau bisa sedikit

akan sangat sulit dalam mendefinisikan nama sebanyak yang dibutuhkan. Maka, dibutuhkan suatu fasilitas untuk mendefinisikan suatu nama kolektif, dan cara untuk mengakses setiap elemen.
Sekumpulan nilai disebut koleksi. Biasanya, suatu koleksi mungkin:

- Kosong (belum mengandung objek/elemen anggota)
- Berisi satu atau lebih objek/anggota

Koleksi dari objek dapat **diorganisasi** dan **diimplementasi** dengan cara tertentu, yang menentukan TYPE kolektif dari objek tsb. TYPE kolektif tersebut selanjutnya diberi nama. Selain cara organisasi, kumpulan objek harus mempunyai aturan dasar operasi terhadap keseluruhan koleksi objek dan terhadap sebuah objek yang merupakan anggota dari koleksi tersebut. Operasi terhadap koleksi objek:

Operasi	**Definisi *domain* dan *range***
Konstruktor untuk Membuat koleksi kosong	→ Koleksi
Selektor, untuk melakukan akses terhadap elemen koleksi objek	Koleksi → elemen
Predikat untuk melakukan test : ▪ apakah koleksi kosong ▪ apakah koleksi masih mampu menampung objek baru (belum penuh)	Koleksi → boolean
Menambahkan sebuah objek ke koleksi objek yang sudah ada (sesuai aturan organisasi)	Elemen, Koleksi → Koleksi
Menghapus sebuah objek dari koleksi (sesuai aturan)	Koleksi → <Koleksi, elemen>
Mengubah nilai sebuah objek tertentu	Koleksi, elemen → Koleksi
Fungsi-fungsi yang mewakili operan dengan koleksi objek sebagai operator (Melakukan operasi terhadap keseluruhan koleksi)	Koleksi → Koleksi
Predikat untuk menentukan apakah sebuah objek tertentu/spesifik ada di antara keseluruhan objek yang ada dalam koleksi *(search, membership)*	Koleksi, elemen → boolean

Operasi	**Definisi *domain* dan *range***
Predikat yang merupakan operator relasional untuk membandingkan dua buah koleksi objek	Koleksi, Koleksi→ boolean

Aturan **organisasi** dari koleksi objek merupakan definisi dari koleksi tersebut.
Beberapa contoh organisasi koleksi objek:

- Tabel (organisasi linier, akses elemen berdasarkan indeks)
- List
- Pohon
- Stack
- Queue
- Graph

Selanjutnya, suatu organisasi objek dapat diimplementasi melalui type dasar yang tersedia dalam paradigma dan bahasanya. Beberapa bahasa sudah menyediakan sarana implementasi secara langsung terhadap koleksi objek. Misalnya :

- Bahasa- bahasa prosedural yang menyediakan tabel (array)
- bahasa LISP yang sudah menyediakan list

TABEL

Definisi

Tabel adalah koleksi objek, wkumpulan elemen yang diroganisasi secara kontigu, dan setiap elemen dapat diakses melalui indeksnya.

Tabel Banyak dijumpai dalam kehidupan sehari-hari, misalnya tabel tarif, perjalanan, jadwal, barang dsb.

Tabel dipakai untuk menguraikan fungsi yang terdefinisi pada interval bilangan bulat [bi... bs] dan memetakan pada harga tertentu. bi ≤ bs

b_i = batas bawah

b_S = batas atas

banyaknya elemen = $b_S - b_i + 1$ disebut ukuran tabel.

Domain harga tabel adalah *product* terhadap type elemen

Konstanta : tabel dituliskan di antara kurung siku, dan setiap harga elemen dipisah dengan koma.

Contoh : [31,28,31,80]

adalah konstanta ber-type tabel terdefinisi

atas interval 1..4 dan berdimensi 4.

Seleksi terhadap elemen tabel : dituliskan dengan indeks $T_i$atau T[i]

dengan i adalah indeks dari tabel T

T_{bi+k-1} adalah elemen ke-k dari tabel T.

Perbedaan tabel [...] dengan type komposisi

- elemen tabel bertype sama,
- elemen type komposisi boleh bermacam-macam
- elemen type komposisi telah diketahui dengan pasti dan fixed pada pendefiiniannya, sedangkan banyaknya elemen tabel sesuai dengan ukurannya yang dapat berupa parameter
- tiap elemen type komposisi bisa diacu dari namanya (dari definisinya) sedangkan elemen tabel dapat diacu dari posisi (indeks) nya sebagai parameter
- tabel adalah koleksi objek, type komposisi adalah sebuah objek

Perbedaan tabel [...] dengan fungsi Selektor F(.)

Perbedaan tabel dengan fungsi akses hanya dari cara penulisan indeks pengakses dan realisasinya oleh sistem. Jika akses langsung dapat disediakan, maka sangat praktis menggunakan tabel. Sebenarnya, penulisan indeks adalah cara penulisan lain dari Selektor terhadap elemen.

Konsekuensi : akses terhadap tabel direalisasi oleh fungsi terdefinisi dalam bahasanya. Perhatikan bahwa dalam bagian LISP, pada akhirnya tabel direpresentasi sebagai list dan dipakai fungsi akses yang didefinisikan berdasarkan primirif akses terhadap list yang merupakan type dasar bahasa LISP. Namun demikian, beberapa versi LISP juga menyediakan fasilitas untuk mendefinisikan tabel *(array)*.

Contoh 1: Pemakaian tabel :

Persoalan : Harike dari suatu tanggal.

Tuliskanlah sebuah fungsi yang menerima sebuah tanggal <d,m,y> dan menghasilkan Harike pada tanggal tersebut, terhitung mulai 1 Januari tahun yang bersangkutan. Contoh masukan dan hasil :

<1,1,1993> $\rightarrow$ 1
<4,2,1979> $\rightarrow$ 35
<31,12,1935> $\rightarrow$ 365

Fungsi Koreksi dapat direalisasi dengan dua cara :

- *Comprehension* (definisi)
- *Extension* (enumerasi)

Dengan anggapan bahwa setiap bulan pasti terdiri dari 31 hari, maka harus dilakukan koreksi terhadap setiap awal bulan. Enumerasi Koreksi pada bulan ke B adalah:

Tabel Koreksi (dengan indeks tabel i adalah bulan ke-i):

0	1	-1	0	0	1	1	2	3	3	4	4
1	2	3	4	5	6	7	8	9	10	11	12

Tabel tsb diturunkan dari tabel sebagai berikut, yang didasari oleh perhitungan awal jika semua bulan dianggap 30 hari :

BULAN	JUMLAH HARI	KOREKSI
1	1	0
2	32	1 + 1x30 + 1
3	60	1 + 2x30 + (-1)
4	91	1 + 3x30 + 0
5	121	1 + 4x30 + 0
6	152	1 + 5x30 + 1
7	182	1 + 6x30 + 1
8	213	1 + 7x30 + 2
9	244	1 + 8x30 + 3
10	274	1 + 9x30 + 3
11	305	1 + 10x30 + 4
12	335	1 + 11x30 + 4