

**DIKTAT KULIAH**  
**DASAR PEMROGRAMAN**  
**Bagian : Pemrograman Fungsional**

Oleh :  
**Inggriani Liem**



**Kelompok Keahlian Data & Rekayasa Perangkat Lunak**  
**Sekolah Teknik Elektro dan Informatika**  
**Institut Teknologi Bandung**  
**Edisi Februari 2008**

# Kata Pengantar

Cikal bakal diktat ini dipakai di lingkungan Jurusan Teknik Informatika ITB sebagai penunjang Kuliah Pemrograman Non Prosedural (pada kurikulum 1993), dan kemudian menjadi kuliah Dasar Pemrograman (IF221 pada kurikulum 1998 dan 2003). Diktat Februari 2008 ini diterbitkan untuk menunjang kuliah Dasar Pemrograman bagi mahasiswa Sekolah Teknik Elektro dan Informatika ITB.

Mahasiswa informatika harus mendapatkan pengetahuan teoritis dan praktek pemrograman dalam beberapa paradigma agar sudut pandang mahasiswa tidak sempit. Pemrograman fungsional merupakan salah satu paradigma yang wajib diberikan. Karena ada kaitannya dengan pengajaran pemrograman pada paradigma lain, diktat ini merupakan salah satu dari seri diktat pemrograman yang menunjang perkuliahan pemrograman program studi Informatika. Beberapa bagian yang dipandang terlalu sulit untuk dibahas karena kurangnya waktu, dapat dibahas pada pelajaran pemrograman dengan paradigma yang berikutnya diajarkan.

Pemrograman Fungsional merupakan paradigma pemrograman yang dipandang cocok untuk diberikan sebagai paradigma pertama, karena mahasiswa belajar abstraksi data dan proses komputasi tanpa perlu memusingkan implementasi fisik memori dan mekanisme eksekusi. Mahasiswa belajar memrogram melalui definisi dan spesifikasi, yang dapat diproses oleh interpreter sederhana. Di Program Studi Teknik Informatika, konsep abstraksi data yang diajarkan pada pemrograman fungsional ini akan diimplementasi lebih lanjut pada pemrograman prosedural maupun berorientasi objek.

Seperti pada paradigma lainnya, pengajaran pemrograman yang disampaikan lewat diktat ini adalah pengajaran konsep yang langsung dilengkapi dengan contoh tipikal berupa contoh program kecil yang utuh yang akan menjadi “pola program” dalam kehidupan nyata sebagai “*programmer*”. Contoh program tersebut mengandung penjelasan-penjelasan yang dituliskan dalam potongan programnya, dan dituliskan dalam notasi fungsional. Program kecil yang ada dan dimuat dalam versi ini merupakan contoh-contoh yang berhasil dikumpulkan, diseleksi, dikompilasi sejak tahun 1993 dari literatur.

Diktat ini dirancang bukan merupakan *text book*, melainkan sebagai buku kerja mahasiswa, yang digunakan untuk membantu mahasiswa dalam memahami dan berlatih merancang, mengkonstruksi, menulis dan membaca program dalam notasi fungsional. Beberapa contoh akan dibahas secara rinci di kelas, sisanya dipakai sebagai acuan dan bahan latihan membaca dan memahami program.

Karena dirancang sebagai buku kerja mahasiswa, mungkin diktat ini sulit untuk dipakai sebagai pegangan bagi pengajar. Rencananya, diktat ini akan dilengkapi dengan buku pedagogi yang ditujukan khusus sebagai pedoman bagi pengajar.

Notasi yang dipakai adalah notasi fungsional. Karena notasi fungsional tidak mempunyai eksekutor, mahasiswa harus mempunyai alat eksekusi dalam bahasa riil dan disadari bahwa akan sulit mendapatkan teks yang bebas kesalahan.

Untuk latihan eksekusi, diktat ini dilengkapi dengan diktat lain yang berisi pedoman penerjemahan ke salah satu bahasa fungsional yang dipilih. Teks program pada diktat ini masih mungkin mengandung kesalahan, walaupun sudah diperbaiki sejak versi pertama tahun 1992, masih ada kesalahan karena ditambahkan program-program baru. Penulis berharap agar kesalahan dapat disampaikan agar dapat diperbaiki pada versi berikutnya.

# DAFTAR ISI

## ***Bagian I : Pemrograman Fungsional***

PENDAHULUAN .....	4
Pradigma Pemrograman .....	4
Bahasa Pemrograman.....	6
Belajar Memprogram Tidak Sama Dengan Belajar Bahasa Pemrograman.....	6
Tujuan Kuliah Pemrograman Fungsional .....	8
Ikhtisar Diktat.....	8
Daftar Pustaka : .....	10
NOTASI FUNGSIONAL .....	11
EKSPRESI DASAR.....	15
Evaluasi Fungsi .....	22
Ekspresi Bernama, Nama Antara .....	23
EKSPRESI KONDISIONAL.....	26
Operator Boolean <u>AND then</u> , <u>OR else</u> .....	27
TYPE BENTUKAN.....	35
Fungsi dengan <i>range</i> type bentukan tanpa nama .....	43
KOLEKSI OBJEK .....	46
TABEL.....	48
EKSPRESI REKURSIF .....	53
Fungsi Rekursif.....	56
Ekspresi Rekursif terhadap Bilangan Bulat .....	59
LIST .....	63
List Dengan Elemen Sederhana .....	64
List of Character (Teks).....	75
List of Integer.....	79
Himpunan (Set).....	86
List of List.....	94
Resume dari analisa rekurens terhadap list .....	101
Resume dari analisa rekurens terhadap list .....	101
POHON .....	102
Pohon N-aire .....	105
Pohon Biner.....	107
Binary Search Tree.....	115
Pohon Seimbang ( <i>balanced tree</i> ).....	116
EKSPRESI LAMBDA.....	117
Fungsi sebagai domain dari fungsi (parameter).....	117
Fungsi Sebagai Hasil Dari Evaluasi (Range).....	121

# PENDAHULUAN

## Pradigma Pemrograman

Komputer digunakan sebagai alat bantu penyelesaian suatu persoalan. Masalahnya, problematika itu tidak dapat "disodorkan" begitu saja ke depan komputer, dan komputer akan memberikan jawabannya. Ada "jarak" antara persoalan dengan komputer. Strategi pemecahan masalah masih harus ditanamkan ke komputer oleh manusia dalam bentuk program. Untuk menghasilkan suatu program, seseorang dapat memakai berbagai pendekatan yang dalam bidang pemrograman disebut sebagai paradigma. Namun demikian, semua pemrograman mempunyai dasar yang sama. Karena itu pada kuliah Dasar pemrograman, diajarkan semua komponen yang perlu dalam pemrograman apapun, walaupun implementasi dan cara konstruksinya akan sangat tergantung kepada paradigma dan bahasa pemrogramannya.

Paradigma adalah sudut pandang atau "sudut serang" tertentu yang diprioritaskan, terhadap kelompok problema, realitas, keadaan, dan sebagainya. Paradigma membatasi dan mengkondisikan jalan berpikir kita, mengarahkan kita terhadap beberapa atribut dan membuat kita mengabaikan atribut yang lain. Karena itu, jelas bahwa sebuah paradigma hanya memberikan pandangan yang terbatas terhadap sebuah realitas. Akibatnya, fanatisme terhadap sebuah paradigma mempersempit wawasan dan bahkan berbahaya.

Dalam pemrograman pun ada beberapa paradigma, masing-masing mempunyai prioritas strategi analisa yang khusus untuk memecahkan persoalan, masing-masing menggiring kita ke suatu pendekatan khusus dari problematika keseluruhan. Beberapa jenis persoalan dapat dipecahkan dengan baik dengan menggunakan sebuah paradigma, sedangkan yang lain tidak cocok. Mengharuskan seseorang memecahkan persoalan hanya dengan melalui sebuah paradigma, berarti membatasi strateginya dalam pemrograman. Satu paradigma tidak akan cocok untuk semua kelas persoalan.

"Ilmu" pemrograman berkembang, menggantikan "seni" memprogram atau memprogram secara coba-coba ("*trial and error*"). Program harus dihasilkan dari proses pemahaman permasalahan, analisis, sintesis dan dituangkan menjadi kode dalam bahasa komputer secara sistematis dan metodologis.

Karena terbatasnya waktu, maka tidak mungkin semua paradigma disatukan dalam sebuah mata kuliah. Pada matakuliah IF221 Dasar Pemrograman, mahasiswa belajar memprogram dengan paradigma fungsional. Paradigma fungsional dapat diajarkan sebagai paradigma pertama, karena sifatnya yang sangat mendasar, bebas memori.

Berikut ini setiap paradigma akan dibahas secara ringkas. Sebenarnya seseorang dapat belajar memprogram dalam salah satu paradigma dan kemudian beranjak ke paradigma lain dengan memakai paradigma yang dipelajarinya sebagai "meta paradigma".

Paradigma tertua dan relatif banyak dipakai saat ini adalah paradigma prosedural. Bahasa fungsional banyak pula diwarnai fasilitas prosedural karena memang ternyata untuk beberapa hal kita belum bisa dari paradigma prosedural akibat mesin pengeksekusi (mesin Von Newmann) yang masih berlandaskan paradigma ini. Dengan alasan ini maka salah satu bagian dari buku ini akan mencakup aspek prosedural dalam paradigma pemrograman fungsional.

Beberapa paradigma dalam pemrograman :

1. Paradigma pemrograman **Prosedural atau imperatif**. Paradigma ini didasari oleh konsep mesin Von Newmann (stored program concept): sekelompok tempat penyimpanan (**memori**), yang dibedakan menjadi memori **instruksi** dan memori **data**; masing-masing dapat diberi nama dan harga. Instruksi akan dieksekusi satu per satu

secara sekuensial oleh sebuah pemroses tunggal. Beberapa instruksi menentukan instruksi berikutnya yang akan dieksekusi (percabangan kondisional). Data diperiksa dan dimodifikasi secara sekuensial pula. Program dalam paradigma ini didasari pada **strukturasi informasi** di dalam memori dan **manipulasi** dari informasi yang disimpan tersebut. Kata kunci yang sering didengungkan dalam pendekatan ini adalah :

**Algoritma + Struktur Data = Program.**

Pemrograman dengan paradigma ini sangat tidak "manusiawi" dan tidak "alamiah", karena harus berpikir dalam batasan mesin (komputer), bahkan kadang-kadang batasan ini lebih mengikat daripada batasan problematiknya sendiri.

Keuntungan pemrograman dengan paradigma ini adalah efisiensi eksekusi, karena dekat dengan mesin.

Pada kurikulum Jurusan Teknik Informatika ITB, pemrograman prosedural dicakup dalam kuliah Algoritma dan Pemrograman dan Struktur Data.

**2. Paradigma pemrograman Fungsional**

Paradigma ini didasari oleh konsep pemetaan dan **fungsi** pada matematika. Fungsi dapat berbentuk sebagai fungsi "primitif", atau komposisi dari fungsi-fungsi lain yang telah terdefinisi. Pemrogram mengasumsikan bahwa ada fungsi-fungsi dasar yang dapat dilakukan. Penyelesaian masalah didasari atas aplikasi dari fungsi-fungsi tersebut. Jadi dasar pemecahan persoalan adalah **transformatif**. Semua kelakuan program adalah suatu rantai transformasi dari sebuah keadaan awal menuju ke suatu rantai keadaan akhir, yang mungkin melalui keadaan antara, melalui **aplikasi** fungsi.

Paradigma fungsional tidak lagi mempernasalahkan memorsasi dan struktur data, tidak ada pemilahan antara data dan program, tidak ada lagi pengertian tentang "variabel". Pemrogram tidak perlu lagi mengetahui bagaimana mesin mengeksekusi atau bagaimana informasi disimpan dalam memori, setiap fungsi adalah "kotak hitam", yang menjadi perhatiannya hanya keadaan awal dan akhir. Dengan merakit kotak hitam ini, pemrogram akan menghasilkan program besar.

Berlainan sekali dengan paradigma prosedural, program fungsional harus diolah lebih dari program prosedural (oleh pemroses bahasanya), karena itu salah satu keberatan adalah kinerja dan efisiensinya.

**3. Paradigma pemrograman Deklaratif, predikatif atau logik**

Paradigma ini didasari oleh pendefinisian **relasi** antar individu yang dinyatakan sebagai **predikat orde pertama**. Sebuah program logik adalah kumpulan aksioma (fakta dan aturan deduksi). Pemrogram mendeklarasikan sekumpulan fakta dan aturan-aturan (*inference rules*). Ketika program dieksekusi, pemakai mengajukan pertanyaan (*Query*), dan program akan menjawab apakah pernyataan itu dapat dideduksi dari aturan dan fakta yang ada. Program akan memakai aturan deduksi dan mencocokkan pertanyaan dengan fakta-fakta yang ada untuk menjawab pertanyaan. Pada kurikulum Jurusan teknik Informatika ITB, pemrograman dengan paradigma ini menjadi bagian dari kuliah Logika Informatika.

**4. Paradigma Pemrograman Berorientasi Objek (*Object Oriented*)**

Paradigma ini didasari oleh **Kelas dan objek**. Objek adalah instansiasi dari kelas. Objek mempunyai atribut (kumpulan sifat), dan mempunyai kelakuan (kumpulan reaksi, metoda). Objek yang satu dapat berkomunikasi dengan objek yang lain lewat "pesan", dengan tetap terjaga integritasnya. Kelas mempunyai hirarki, anggota dari

sebuah kelas juga mendapatkan turunan atribut dari kelas di atasnya. Paradigma ini menawarkan konsep modularitas, penggunaan kembali, kemudahan modifikasi.

Dalam paradigma ini, masih terkandung dari paradigma imperatif, karena mengkonstruksi program dari objek dan kelas adalah tidak berbeda dengan mengkonstruksi program dari struktur data dan algoritma, dengan cara enkapsulasi menjadi kelas. Kedekatan antara paradigma ini dengan paradigma lain dapat dilihat dari bahasa-bahasa bukan berorientasi obyek murni, yaitu bahasa prosedural atau fungsional yang ditambahi dengan ciri orientasi objek.

#### 4. **Paradigma Konkuren**

Paradigma ini didasari oleh kenyataan bahwa dalam keadaan nyata, sebuah sistem komputer harus menangani beberapa proses (*task*) yang harus dieksekusi bersama dalam sebuah lingkungan (baik mono atau multi prosesor). Pada pemrograman konkuren, pemrogram tidak lagi berpikir sekuensial, melainkan harus menangani komunikasi dan sinkronisasi antar task. Pada jaman sekarang, aspek konkuren semakin memegang peranan penting dan beberapa bahasa menyediakan mekanisme dan fasilitas yang mempermudah implementasi program konkuren.

Salah satu aspek penting dalam pemrograman berorientasi objek dan merupakan topik pemrograman yang “lanjut” adalah bagaimana menangani objek yang “hidup bersama, secara konkuren”. Maka pada kurikulum Jurusan Informatika, paradigma pemrograman konkuren dan berorientasi Objek dicakup dan disatukan dalam matakuliah Pemrograman Objek Konkuren.

## **Bahasa Pemrograman**

Berbicara mengenai bahasa pemrograman, ada banyak sekali bahasa pemrograman, mulai dari bahasa tingkat rendah (bahasa mesin dalam biner), bahasa assembler (dalam kode mnemonik), bahasa tingkat tinggi, sampai bahasa generasi ke empat (4GL).

Bahasa Pemrograman berkembang dengan cepat sejak tahun enam puluhan, seringkali dianalogikan dengan menara Babel yang berakibat manusia menjadi tidak lagi saling mengerti bahasa masing-masing. Untuk setiap paradigma, tersedia bahasa pemrograman yang mempermudah implementasi rancangan penyelesaian masalahnya. Contoh bahasa-bahasa pemrograman yang ada :

1. Prosedural : Algol, Pascal, Fortran, Basic, Cobol, C , Ada
2. Fungsional : LOGO, APL, LISP
3. Deklaratif : Prolog
4. Object oriented murni: Smalltalk, Eiffel, Java.

Pemroses bahasa Pascal dan C versi terbaru dilengkapi dengan fasilitas orientasi objek, misalnya Turbo Pascal (mulai versi 5.5) pada PC dan C++. Beberapa fasilitas “packaging” dan inheritance yang disediakan bahasanya seperti dalam bahasa Ada memungkinkan implementasi program secara berorientasi objek secara lebih natural.

## **Belajar Memprogram Tidak Sama Dengan Belajar Bahasa Pemrograman**

Belajar memprogram adalah belajar tentang strategi pemecahan masalah, metodologi dan sistematika pemecahan masalah tersebut kemudian menuangkannya dalam suatu notasi yang disepakati bersama. Beberapa masalah akan cocok kalau diselesaikan dengan suatu paradigma tertentu. Karena itu, pengetahuan tentang kelas persoalan penting adanya.

Pada hakekatnya, penggunaan komputer untuk memecahkan persoalan adalah untuk tidak mengulang-ulang kembali hal yang sama. Strategi pengenalan masalah melalui

dekomposisi, pemakaian kembali modul yang ada, sintesa, selalu dipakai untuk semua pendekatan, dan seharusnya mendasari semua pengajaran pemrograman. Karena itu perlu diajarkan metodologi, terutama karena sebagian besar pemrogram pada akhirnya memakai program yang sudah pernah ditulis orang lain dibandingkan dengan bongkar pasang program yang sudah ada.

Belajar memprogram lebih bersifat pemahaman persoalan, analisis, sintesis.

Belajar bahasa pemrograman adalah belajar memakai suatu bahasa, aturan sintaks (tatabahasa), setiap instruksi yang ada dan tata cara pengoperasian kompilator atau interpreter bahasa yang bersangkutan pada mesin tertentu. Lebih lanjut, belajar bahasa pemrograman adalah belajar untuk memanfaatkan instruksi-instruksi dan kiat yang dapat dipakai secara spesifik hanya pada bahasa itu. Belajar memprogram lebih bersifat keterampilan dari pada analisis dan sintesa.

Belajar memprogram dan belajar bahasa pemrograman mempunyai tingkatan dan kesulitan yang berbeda-beda. Mahasiswa seringkali dihadapkan pada kedua kesulitan itu sekaligus. Pemecahan persoalan dengan paradigma yang sama akan menghasilkan solusi yang "sejenis". Beberapa bahasa dapat termasuk dalam sebuah paradigma sama, pemecahan persoalan dalam satu paradigma dapat diterjemahkan ke dalam bahasa-bahasa yang berbeda. Untuk itulah diperlukan adanya suatu perjanjian, notasi yang disepakati supaya masalah itu dapat dengan mudah diterjemahkan ke dalam salah satu bahasa yang masih ada dalam lingkup paradigma yang sama. Pada pengajaran pemrograman fungsional ini dikembangkan suatu notasi yang disebut notasi fungsional, yang akan dipakai sebagai "bahasa ekspresi dan komunikasi" antara persoalan dengan pemrogram,

Pemilihan paradigma dan strategi pemecahan persoalan lebih penting daripada pemilihan bahasanya sendiri, walaupun pada akhirnya memang harus diputuskan bahasa yang dipakai. Bahasa pemrograman fungsional yang paling banyak dipakai saat ini adalah bahasa yang berinduk pada LISP. Karena itu pada bagian akhir buku ini diberikan contoh terjemahan dari notasi fungsional menjadi program dalam bahasa LISP atau salah satu dialeknya.

Proses memprogram adalah proses yang memerlukan kepakaran, proses koding lebih merupakan proses semi otomatis dengan aturan pengkodean. Proses memprogram memang berakhir secara konkrit dalam bentuk program yang ditulis dan dieksekusi dalam bahasa target. Karena itu memaksa mahasiswa hanya bekerja atas kertas, menganalisis dan membuat spesifikasi tanpa pernah me-run program adalah tidak benar. Sebaliknya, hanya mencetak pemrogram yang langsung "memainkan keyboard", mengetik program dan mengeksekusi tanpa analisis dan spesifikasi yang dapat dipertanggung jawabkan juga tidak benar (terutama untuk program skala besar dan harus dikerjakan banyak orang).

Produk yang dihasilkan oleh seorang pemrogram adalah program dengan rancangan yang baik (metodologis, sistematis), yang dapat dieksekusi oleh mesin, berfungsi dengan benar, sanggup melayani segala kemungkinan masukan, dan didukung dengan adanya dokumentasi. Pengajaran pemrograman titik beratnya adalah membentuk seorang perancang "*designer*" program, sedangkan pengajaran bahasa pemrograman titik beratnya adalah membentuk seorang "*coder*" (juru kode).

Pada prakteknya, suatu rancangan harus dapat dikode untuk dieksekusi dengan mesin. Karena itu, belajar pemrograman dan belajar bahasa pemrograman saling komplementer, tidak mungkin dipisahkan satu sama lain.

Metoda terbaik untuk belajar apapun adalah melalui contoh. Seorang yang sedang belajar harus belajar melalui contoh nyata. Berkat contoh nyata itu dia melihat, mengalami dan melakukannya. Metoda pengajaran yang dipakai pada perkuliahan pemrograman fungsional ini adalah pengajaran melalui contoh tipikal. Contoh tipikal adalah contoh

program yang merupakan “pola solusi” dari kelas-kelas persoalan yang dapat diselesaikan dengan paradigma pemrograman fungsional.

## **Tujuan Kuliah Pemrograman Fungsional**

Tujuan utama dari matakuliah ini adalah membekali mahasiswa cara berpikir dan pemecahan persoalan dalam paradigma pemrograman fungsional. Mahasiswa harus mampu membuat penyelesaian masalah pemrograman fungsional tanpa tergantung pada bahasa pemrograman apapun, dan kemudian ia mampu untuk mengeksekusi programnya dengan salah satu bahasa pemrograman fungsional LISP. Mahasiswa akan memakai bahasa pemrograman tersebut sebagai alat untuk mengeksekusi program dengan mesin yang tersedia.

Secara lebih spesifik, mahasiswa diharapkan mampu untuk :

1. Memecahkan masalah dengan paradigma fungsional tanpa tergantung pada bahasa pemrograman apapun.
2. Menulis program berdasarkan pemrograman fungsional menjadi salah satu bahasa pemrograman yang menjadi target (misalnya LISP).

Ada beberapa alasan, mengapa kuliah Dasar Pemrograman diisi dengan pemrograman fungsional:

1. Pada hakekatnya, program dibuat untuk melaksanakan suatu fungsi tertentu sesuai dengan kebutuhan pemakai. Jika sebuah fungsi dapat kita umpamakan sebuah tombol khusus pada “mesin” komputer, maka mengaktifkan program untuk pemecahan persoalan idealnya adalah hanya dengan menekan sebuah tombol saja.
2. Pada pemrograman fungsional, kita dihadapkan kepada cara berpikir melalui fungsi (apa yang akan direalisasikan) tanpa mempedulikan bagaimana memori komputer dialokasi, diorganisasi, diimplementasi tanpa menghiraukan sekuens/urutan instruksi karena sebuah program hanyalah aplikasi terhadap sebuah fungsi
3. Pada paradigma fungsional, kita juga “terbebas” dari persoalan eksekusi program, karena eksekusi program hanyalah aplikasi terhadap sebuah fungsi.

## **Ikhtisar Diktat**

Diktat ini terdiri dari dua bagian

**Bagian I – Notasi Fungsional** difokuskan kepada pemrograman fungsional dengan kasus-kasus tipikal serta primitif yang berguna untuk memprogram dalam skala lebih besar. Dapat dikatakan bahwa bagian pertama ini berisi nukleus dan atom pembentuk aplikasi nyata dalam program fungsional.

Bagian pertama buku ini secara garis besar akan berisi : (akan diuraikan lebih detail)

1. Pendahuluan : pengenalan akan paradigma pemrograman, dan pelajaran pemrograman serta cakupan dari diktat ini
2. Konsep dasar dan notasi fungsional
3. Ekspresi dasar dan evaluasi fungsi
4. Ekspresi kondisional
5. Type bentukan (objek) dalam konteks fungsional. Contoh tipikal akan disajikan melalui tiga type dengan komponen dasar yang sering dipakai dalam informatika: Point, Pecahan dan Date. Melalui contoh tersebut, diharapkan mahasiswa mampu memperluas aplikasinya menjadi type bentukan yang lain, bahkan membentuk type bentukan dari type bentukan..
6. **Koleksi objek**, yang mendasari organisasi dan struktur data. Hanya diberikan sebagai introduksi

7. **Tabel.** Pada abagian ini akan diajarkan bagaimana realisasi tabel yang secara konsep banyak dipakai dalam informatika dalam konsep fungsional, yaitu sebagai tabulasi eksplisit atau berdasarkan fungsi
8. **Ekspresi rekursif.** Ekspresi rekurens yang dibangun berdasarkan Analisa rekurens adalah salah satu landasan berpikir dalam pemrograman fungsional yang sangat penting. Bagian ini menjadi dasar dari bagian-bagian selanjutnya, dan kira-kira akan memakan porsi hampir dari separuh jam kuliah yang dialokasikan. Sebagai contoh permulaan, akan diberikan **Ekspresi rekursif terhadap bilangan integer.**
9. **List:** sebuah type data rekursif yang mewakili koleksi objek. List adalah type data dasar yang banyak dipakai dalam informatika, dan khususnya menjadi struktur data dasar dalam bahasa LISP, sehingga hampir semua persoalan yang diselesaikan dalam LISP akan didasari oleh struktur data list ini. Pada bagian ini akan dicakup:
  - 9.1. List dengan elemen sederhana
  - 9.2. List dengan elemen karakter (teks)
  - 9.3. List dengan elemen bilangan integer
  - 9.4. Himpunan (Set), yaitu list dengan elemen yang harus unik
  - 9.5. List dengan elemen list (list of list)
10. **Pohon** dan contoh kasusnya. Beberapa representasi pohon, khususnya pohon biner akan dibahas pada bagian ini yaitu prefix, infix dan postfix. Kasus yang dibahas adalah evaluasi ekspresi yang representasi internalnya adalah pohon.
11. **Ekspresi Lambda**, bagian terakhir ini membahas tentang konsep paling rumit dalam pemrograman fungsional, dimana dalam definisi pemetaan fungsional, sekarang **domain** dan **range** dari fungsi adalah fungsi dan bukan lagi merupakan type biasa.

**Buku II – LISP** difokuskan pada penulisan program fungsional dalam bahasa LISP. Selain membahas LISP secara ringkas dan pola translasi dari notasi fungsional ke LISP, pada bagian ini semua konsep yang ditulis di bagian pertama akan diberikan aturan translasinya. Pada bagian kedua ini juga dicakup aspek eksekusi program LISP: mahasiswa belajar memahami aspek eksekusi (evaluasi fungsional) melalui contoh-contoh yang diberikan, yang sudah dicoba dengan kompilator CGLISP. Menyadari bahwa dalam keluarga bahasa LISP banyak “varian” nya, maka diusahakan, agar instruksi yang dipakai hanyalah instruksi standard.

Beberapa fungsi yang dituliskan dalam bagian pertama sengaja diulang penulisannya, untuk mempermudah pembaca dalam membaca. Tidak semua fungsi dasar dan contoh yang diberikan pada bagian kesatu diterjemahkan dalam bagian kedua. Hal ini sengaja dilakukan agar mahasiswa berlatih sendiri dengan mesin untuk menterjemahkannya, karena sekarang mahasiswa sudah mempunyai mesin pengeksekusi..

**Daftar Pustaka :**

1. Abelson H., Sussman G.J., and Sussman J. : "Structure and Interpretation of Computer Programs", MIT Press, McGraw-Hill, 4th printing, 1986.
2. Bird R. and Wadler P. : "An Introduction to Functional Programming", Prentice-Hall International, 1988.
3. Scholl P.C., Fauvet M.C., Lagnier F., Maraninchi F. : "Cours d'informatique: langage et programmation", Masson (Paris), 1993.
4. Friedman D. and Felleisen M. : "The Little LISPer", Pergamon Pub.Co., 3rd edition, 1989.
5. Steele G.L. : "Common LISP", 1984.
6. Winston P.H. and Horn B.K.P. : "LISP", Addison-Wesley, 3rd edition, 1989.

## NOTASI FUNGSIONAL

Sebuah program komputer adalah “model”, yang mewakili solusi persoalan tertentu dalam informatik yang akan diselesaikan dengan komputer. Program berisi kumpulan informasi penting yang mewakili persoalan itu.

Pada paradigma pemrograman fungsional solusi persoalan diungkapkan menjadi identifikasi dari satu atau beberapa fungsi, yang jika di "aplikasi" dengan nilai yang diberikan akan memberikan hasil yang diharapkan. Dalam paradigma fungsional, program direpresentasi dalam: himpunan nilai type, dengan nilai-nilai dari type adalah konstanta.

Fungsi adalah asosiasi (pemetaan) antara dua type yaitu *domain* dan *range*, yang dapat berupa :

- type dasar
- type terkomposisi (bentukan)

Untuk menuliskan suatu program fungsional, dipakai suatu bahasa ekspresi .

Ada tiga macam bentuk komposisi ekspresi :

- ekspresi fungsional dasar
- kondisional
- rekursif

Masing-masing ekspresi akan dibahas pada bagian selanjutnya.

Untuk sebagian besar pembaca yang sudah memahami bagaimana memprogram secara prosedural, berikut ini diberikan ilustrasi mengenai perbedaan antara program fungsional dengan program imperatif (prosedural). Bagi yang belum pernah mempelajari program prosedural, bagian ini dapat diloncati tanpa mengubah alur pemahaman

Perhatikan sebuah program yang ditulis dalam bahasa algoritmik sebagai berikut :

<b>PROGRAM PLUSAB</b> { Membaca dua buah nilai a dan b integer, menghitung jumlahnya dan menuliskan hasilnya }
<u>Kamus</u> : a,b : integer
<u>Algoritma</u> :  <b>input</b> (a,b) <b>output</b> (a+b)

Program tersebut mengandung instruksi pembacaan nilai (input) dan penulisan hasil (output). Program akan menunggu aksi pembacaan dilakukan, melakukan kalkulasi dan akan mencetak hasil. Ada suatu sekuens (urut-urutan) aksi yang dilakukan.

Kelakuan (*behaviour*) dari program fungsional berbeda. Dalam pemrograman Fungsional tidak ada 'aksi' menggunakan baca/tulis atau mengubah *state*.

Pada konsteks fungsional, ekivalensi dari program diatas adalah pemakai mengetik 3+4 sistem menghasilkan 7. Semua yang dilakukan pemakai ini telah mewakili aksi baca/tulis pada program "aksional" (berdasarkan "aksi", *action*)

<b>APLIKASI</b>
⇒ 3+4
⇒ 7

Pemrograman fungsional didasari atas analisa *top down*:

1. Problema
2. Spesifikasi
3. Dekomposisi pada persoalan "antara", berarti menciptakan sebuah fungsi antara

**Fungsi pada analisa *topdown* adalah** strukturasi teks. Sebuah fungsi mewakili sebuah tingkatan abstraksi. Dengan mengenalkan (mendefinisikan) sebuah fungsi, maka pemrogram memperkaya "khasanah", perbendaharaan dari fungsi yang tersedia, dan karena sudah didaftarkan maka dapat digunakan kemudian.

### Konstruksi program fungsional : definisi-spesifikasi-realisisi-aplikasi

<b>Tahapan</b>	<b>Deskripsi</b>
<b>Definisi</b>	Kata kuncinya adalah <b>memberikan identitas fungsi, yaitu menentukan nama, domain dan range.</b> Contoh: untuk memangkatkan sebuah bilangan integer dengan tiga, didefinisikan nama Pangkat3, dengan domain integer dan range adalah integer. Dituliskan <b>Pangkat3 : integer → integer</b>
<b>Spesifikasi</b>	Kata kuncinya adalah menentukan "apa" yang dilakukan oleh fungsi, yaitu menentukan 'arti' dari fungsi. Contoh : Fungsi bernama Pangkat3(x) artinya menghitung pangkat tiga dari nilai x .
<b>Realisasi</b>	Kata kuncinya adalah menentukan "bagaimana" fungsi melakukan komputasi, yaitu mengasosiasikan pada nama fungsi, sebuah <b>ekspresi fungsional</b> dengan parameter formal yang cocok. Contoh : mengasosiasikan pada Pangkat Tiga : $a*a*a$ atau $a^3$ dengan a adalah nama parameter formal. Parameter formal fungsi adalah nama yang dipilih untuk mengasosiasikan domain dan range.
<b>Aplikasi</b>	Adalah memakai fungsi untuk melakukan komputasi, atau memakainya dalam suatu ekspresi, yaitu dengan menggantikan semua nama parameter formal dengan nilai. Dengan aplikasi fungsi, akan dilakukan evaluasi ekspresi fungsional. Contoh : Pangkat Tiga (2) + Pangkat Tiga (3) Argumen pada saat dilakukan aplikasi fungsi disebut parameter aktual

Pada perkuliahan ini dipakai suatu notasi untuk menuliskan program fungsional yang disebut **sebagai notasi fungsional**. Dengan notasi fungsional, teks terdiri dari judul dan empat bagian sesuai dengan tahapan pemrograman di atas.

Bagian pertama adalah “*header*” (judul program), yang berisi Judul Fungsi deskripsi sangat ringkas dari fungsi dan nama serta parameter formalnya. Bagian ini cukup memberikan penjelasan bagi pemakai fungsi andaikata sudah pernah direalisasikan.

Bagian kedua berisi definisi dan spesifikasi fungsi sesuai dengan keterangan di atas.; kedua ini (definisi dan spesifikasi) tidak dipisahkan satu sama lain karena sangat erat kaitannya.

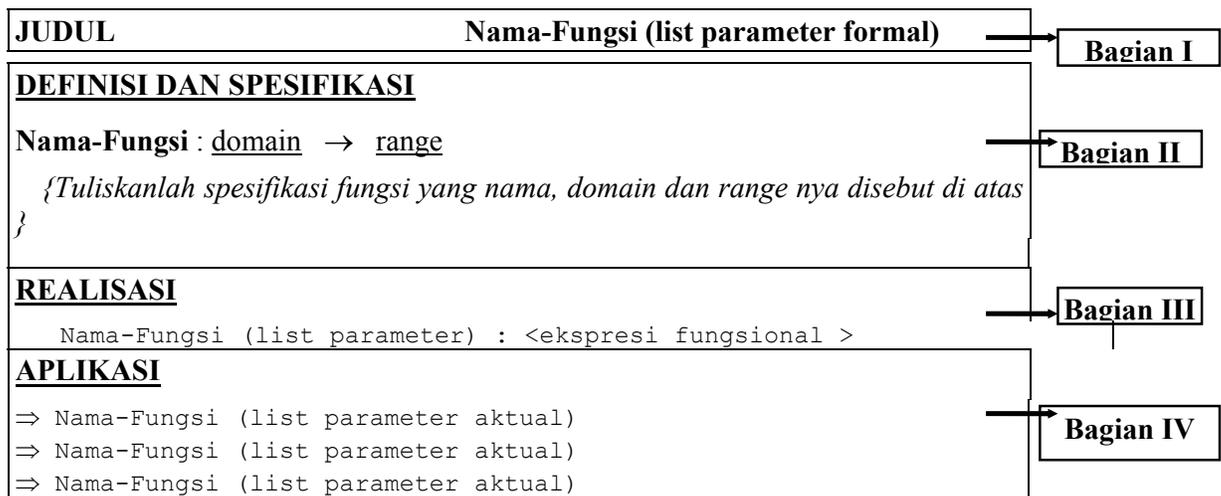
Bagian ketiga berisi realisasi fungsi, yaitu **ekspresi fungsional** yang ditulis untuk mencapai spesifikasi yang dimaksudkan. Sebuah definisi dan spesifikasi yang sama dapat direalisasikan dalam beberapa ekspresi ! Perhatikanlah pula bahwa dalam realisasi fungsi, nama fungsi dituliskan berikut ekspresinya. Tidak diperlukan kata kunci “fungsi” karena dalam konteks fungsional, semua objek adalah fungsi.

Bagian keempat berisi contoh aplikasi fungsi, jika dipandang perlu, dan bahkan hasilnya. Bagian inilah yang sebenarnya akan merupakan interaksi langsung dengan pemakai pada konsteks eksekusi.

Bagian teks yang berupa komentar dituliskan di antara kurung kurawal

Notasi fungsional ini dibuat untuk menuliskan rancangan program supaya lebih mudah dibaca oleh “manusia”.

Berikut ini adalah contoh generik (*template*) teks program dalam notasi fungsional



Lihatlah contoh dari penggunaan notasi ini pada bagian berikutnya Seluruh sisa buku ini akan memakai notasi seperti di atas.

Notasi fungsional ini diadopsi dari [3] dan dikembangkan khusus untuk perkuliahan ini, dan mampu menampung semua konsep pemrograman fungsional, walaupun tidak mungkin dieksekusi (karena tidak mempunyai pemroses bahasa). Kekuatan aspek konseptual, statis dan aspek desain ini justru diperoleh karena ketiadaan dari eksekutor. Perancang dibebaskan dari keterbatasan bahasa, terutama aspek eksekusi yang dinamik. Akibatnya perancang harus berpikir untuk mendapatkan solusi yang secara statik benar.

Karena tidak mempunyai eksekutor, maka notasi ini harus dilengkapi dengan aturan translasi ke bahasa riil yang ada. Buku kedua akan melengkapi notasi ini dengan aturan translasi ke LISP.

Kebanyakan bahasa fungsional mempunyai notasi penulisan yang lebih rumit dan tidak sesuai dengan kebiasaan sehari-hari (misalnya notasi prefix). Penulisan program langsung ke dalam bahasa fungsional terutama bagi pemula akan sangat membingungkan dan berpotensi menimbulkan kesalahan sintaks. Notasi fungsional dibuat untuk mempermudah dalam menentukan solusi. Setelah solusi didapat, translasi ke bahasa fungsional dapat dilakukan secara “mekanistik”.

## EKSPRESI DASAR

Seperti telah dijelaskan pada bagian pendahuluan, pada pemrograman fungsional, pemrogram mulai dari **fungsi dasar** yang disediakan oleh pemroses bahasa untuk membuat fungsi lain yang melakukan aplikasi terhadap fungsi dasar tersebut.

**Fungsi yang paling dasar** pada program fungsional disebut **operator**. Pada ekspresi fungsional, sebagai “titik awal”, dinyatakan bahwa tersedia operator aritmatika, operator relasional dan operator boolean.

Jenis operator	Notasi	Deskripsi
Operator aritmatika	*, /, + - mod div	Berlaku untuk operan numerik. untuk melakukan perhitungan Selain operator aritmatika tsb, diasumsikan pula tersedia fungsi dasar perhitungan bilangan “integer” seperti abs, mod, div. Dengan operator sederhana dan fungsi dasar tersebut, akan diberikan contoh-contoh pengembangan program fungsional yang hanya membutuhkan ekspresi aritmatika tersebut.
Operator relasional	<, >, =, ≤, ≥, ≠	Berlaku untuk operan numerik atau karakter, hasilnya adalah boolean
Operator boolean	<b>and</b> , <b>or</b>	Berlaku untuk operan boolean, sesuai dengan definisi <b>and</b> dan <b>or</b> pada aljabar boolean

Ekspresi adalah sebuah teks yang terdiri dari: nama, simbol, operator, fungsi, ( ), yang dapat menghasilkan suatu nilai berkat evaluasi dari ekspresi.

**Ekspresi aritmatika (numerik), adalah ekspresi dengan operator aritmatika ‘\*’, ‘/’, ‘+’, ‘-’** dapat dituliskan dalam bentuk infix, prefix atau postfix. Pada notasi fungsional, jenis ekspresi yang dipakai adalah Infix, karena lebih manusiawi.

Jenis	Ekspresi aritmatika	Ekspresi boolean
Infix	(3+4) *5	3 < 5
Prefix	(* (+ 3 4) 5)	< 3 5
Postfix	(3 4 + ) 5 *	3 5 >

Ekspresi di atas tidak mengandung nama, melainkan hanya mengandung simbol angka numerik, operator dan tanda kurung

Hasil evaluasi (perhitungan) suatu ekspresi dasar dapat berupa nilai numerik atau boolean. Ekspresi yang hasilnya numerik disebut ekspresi numerik (aritmatika). Ekspresi yang hasilnya boolean disebut ekspresi boolean. Jika sebuah ekspresi boolean operatornya numerik, disebut pula ekspresi relasional.

Selain menggunakan konstanta numerik tersebut, ekspresi dapat berupa ekspresi aljabar :

- abstraksi dengan menggunakan "nama"
- nama mempunyai "nilai"

yang hanya mengandung operator aritmatika.

Evaluasi ekspresi tergantung kepada presedensi (prioritas evaluasi) dan aturan yang ditetapkan.

Contoh :

- $a^3$  : Evaluasi hanya mungkin terjadi jika a diberi nilai (diaplikasi)
- $3+4*5$  dengan aturan presedensi \* yang lebih tinggi dari + maka akan dievaluasi sebagai  $3 + (4*5)$

Fungsi dengan hasil nilai boolean disebut PREDIKAT, dan namanya biasanya diawali dengan “Is”. Contoh: **IsAnA?**: character → boolean yang bernilai benar jika C adalah karakter ‘A’

Berikut ini diberikan contoh-contoh persoalan yang dapat diselesaikan secara fungsional hanya dengan membuat ekspresi dasar fungsional, yaitu dengan operator aritmatika, operator relasional, operator boolean dan fungsi dasar terhadap bilangan seperti mod, div, abs.

Pada hakekatnya, fungsi akan menghasilkan suatu nilai karena di dalam fungsi tsb dilakukan evaluasi ekspresi.

### Contoh-1 Ekspresi numerik : PANGKAT DUA

#### Persoalan :

Buatlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menerima sebuah bilangan bulat dan menghasilkan pangkat dua dari bilangan tersebut dengan menuliskan ekspresi yang hanya mengandung operator ekspresi aritmatika yang telah didefinisikan

<b>PANGKAT2</b>	<b>FX2(x)</b>
<b><u>DEFINISI DAN SPESIFIKASI</u></b>	
<b>FX2</b> : <u>integer</u> → <u>integer</u> <i>{FX2 (x) menghitung pangkat dua dari x, sebuah bilangan integer }</i>	
<b><u>REALISASI</u></b>	
<b>FX2 (x) : x * x</b>	
<b><u>APLIKASI</u></b>	
⇒ FX2 (1) ⇒ FX2 (0) ⇒ FX2 (-1)	

### Contoh-2 Ekspresi numerik: PANGKAT TIGA

#### Persoalan :

Buatlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menerima sebuah bilangan bulat dan menghasilkan pangkat tiga dari bilangan tersebut dengan menuliskan ekspresi yang hanya mengandung operator ekspresi aritmatika yang telah didefinisikan

<b>PANGKAT3 (versi-1)</b>	<b>FX3(x)</b>
<b><u>DEFINISI DAN SPESIFIKASI</u></b>	
<b>FX3</b> : <u>integer</u> → <u>integer</u> <i>{FX3 (x) menghitung pangkat tiga dari x, sebuah bilangan integer }</i>	

<b><u>REALISASI</u></b>
<b>FX3</b> (x) : $x * x * x$
<b><u>APLIKASI</u></b>
⇒ FX3 (1) ⇒ FX3 (8) ⇒ FX3 (-1)

**Contoh-2 Ekspresi numerik: PANGKAT3 (dengan fungsi antara)**

**Persoalan :**

Buatlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menerima sebuah bilangan bulat dan menghasilkan pangkat tiga dari bilangan tersebut dengan menuliskan ekspresi yang hanya mengandung ekspresi perkalian dan aplikasi terhadap fungsi PANGKAT2 yang telah didefinisikan.

<b>PANGKAT3 (versi 2)</b>	<b>FX3(x)</b>
<b><u>DEFINISI DAN SPESIFIKASI</u></b>	
<b>FX3</b> : <u>integer</u> → <u>integer</u> <i>{FX3 (x) menghitung pangkat tiga a dari x, sebuah bilangan integer, dengan aplikasi FX2 sebagai fungsi antara}</i>	
<b>FX2</b> : <u>integer</u> → <u>integer</u> <i>{FX2 (x) menghitung pangkat dua dari x, sebuah bilangan integer }</i>	
<b><u>REALISASI</u></b>	
<b>FX3</b> (x) : $x * FX2(x)$	
<b><u>APLIKASI</u></b>	
⇒ FX3 (8) ⇒ FX3 (1) ⇒ FX3 (-1)	

**Contoh-3 Ekspresi dengan analisa bottom up: realisasi MAX3, jika dipunyai MAX2**

**Persoalan :**

Buatlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menerima tiga buah bilangan bulat dan menghasilkan nilai maksimum dari ketiga bilangan tersebut dengan melakukan aplikasi terhadap MAX2 yang telah didefinisikan.

<b>MAKSIMUM 3 BILANGAN INTEGER</b>	<b>MAX3(a,b,c)</b>
<b><u>DEFINISI DAN SPESIFIKASI</u></b>	
<b>MAX3</b> : 3 <u>integer</u> → <u>integer</u>	



Perhatikan bahwa kalkulasi untuk menghasilkan nilai terbesar dan terkecil dari dua buah bilangan dapat dilakukan dengan suatu ekspresi aritmatika, yang tentunya memerlukan pengetahuan mengenai “rumus” yang dipakai. Sedangkan maksimum dari 4 buah bilangan ditentukan dengan membandingkan nilai maksimum dari dua bilangan, dengan cara mencari maksimum dari maksimum yang diperoleh terhadap dua buah integer. Demikian pula dengan nilai minimum.

MEAN-OLYMPIQUE	MO (u,v,w,x)
<p><b><u>DEFINISI DAN SPESIFIKASI</u></b></p> <p><b>MO</b> : 4 <u>integer</u> <math>\geq 0 \rightarrow</math> <u>real</u>  <i>{ MO (u,v,w,x): menghitung rata-rata dari dua buah bilangan integer, yang bukan maksimum dan bukan minimum dari 4 buah integer: <math>(u+v+w+x-\min_4(u,v,w,x)-\max_4(u,v,w,x))/2</math>}</i></p> <p><b>max4</b> : 4 <u>integer</u> <math>&gt;0 \rightarrow</math> <u>integer</u>  <i>{max4 (i,j,k,l) menentukan maksimum dari 4 buah bilangan integer}</i></p> <p><b>min4</b> : 4 <u>integer</u> <math>&gt; 0 \rightarrow</math> <u>integer</u>  <i>{min4 (i,j,k,l) menentukan minimum dari 4 buah bilangan integer}</i></p> <p><b>max2</b> : 2 <u>integer</u> <math>&gt;0 \rightarrow</math> <u>integer</u>  <i>{max2 (a,b) menentukan maksimum dari 2 bilangan integer, hanya dengan ekspresi aritmatika: jumlah dari kedua bilangan ditambah dengan selisih kedua bilangan, hasilnya dibagi 2}</i></p> <p><b>min2</b> : 2 <u>integer</u> <math>&gt; 0 \rightarrow</math> <u>integer</u>  <i>{min2 (a,b) menentukan minimum dari 2 bilangan integer, hanya dengan ekspresi aritmatika :jumlah dari kedua bilangan – selisih kedua bilangan, hasilnya dibagi 2}</i></p>	
<p><b><u>REALISASI</u></b></p> <p><b>max2</b> (a,b) : <math>(a + b + \text{abs}(a - b)) / 2</math>  <b>min2</b> (a,b) : <math>(a + b - \text{abs}(a - b)) / 2</math>  <b>max4</b> (i, j, k, l) : <math>\text{max2}(\text{max2}(i, j), \text{max2}(k, l))</math>  <b>min4</b> (i, j, k, l) : <math>\text{min2}(\text{min2}(i, j), \text{min2}(k, l))</math></p> <p><b>MO</b> (u, v, w, x) : <math>(u+v+w+x-\min_4(u, v, w, x) - \max_4(u, v, w, x)) / 2</math></p>	
<p><b><u>APLIKASI</u></b></p> <p><math>\Rightarrow</math> MO (8, 12, 10, 20)</p>	

### Contoh-5 Ekspresi boolean : POSITIF

#### Persoalan :

Buatlah definisi, spesifikasi dan realisasi dari sebuah predikat yang menerima sebuah bilangan bulat dan bernilai benar jika bilangan tersebut positif. Lebih spesifik : menghasilkan sebuah nilai boolean yang bernilai true jika bilangan tersebut positif, atau false jika bilangan tersebut negatif.

POSITIF	IsPositif?(x)
<b><u>DEFINISI DAN SPESIFIKASI</u></b> <b>IsPositif?</b> : <u>integer</u> → <u>boolean</u> <i>{IsPositif? (x) benar jika x positif }</i>	
<b><u>REALISASI</u></b> <b>IsPositif?</b> (x) : $x \geq 0$	
<b><u>APLIKASI</u></b> ⇒ IsPositif?(1) ⇒ IsPositif? (0) ⇒ IsPositif? (-1)	

### Contoh-6 Ekspresi boolean : APAKAH HURUF A

#### Persoalan :

Buatlah definisi, spesifikasi dan realisasi dari sebuah predikat yang menerima sebuah karakter dan bernilai benar jika karakter tersebut adalah huruf 'A'.

APAKAH HURUF A	IsAnA?(C)
<b><u>DEFINISI DAN SPESIFIKASI</u></b> <b>IsAnA</b> : <u>character</u> → <u>boolean</u> <i>{IsAnA (C) benar jika c adalah karakter (huruf) 'A' }</i>	
<b><u>REALISASI</u></b> <b>IsAnA?</b> (c) : $c = 'A'$	
<b><u>APLIKASI</u></b> ⇒ IsAnA?(1) ⇒ IsAnA? (0) ⇒ IsAnA? (-1)	

### Contoh-7 Ekspresi boolean : APAKAH ORIGIN

#### Persoalan :

Buatlah definisi, spesifikasi dan realisasi dan contoh aplikasi dari sebuah predikat yang menerima dua buah bilangan riil yang interpretasinya adalah absis dan ordinat pada sumbu kartesian, dan mengirimkan apakah absis dan ordinat tersebut merupakan titik O(0,0)

APAKAH ORIGIN	IsOrigin?(x,y)
<b><u>DEFINISI DAN SPESIFIKASI</u></b>	
<b>IsOrigin?</b> : <u>real</u> , <u>real</u> → <u>boolean</u> <i>{IsOrigin? (x,y) benar jika (x,y) adalah dua nilai yang mewakili titik origin (0,0) }</i>	
<b><u>REALISASI</u></b>	
<b>IsOrigin?</b> (x, y) : x=0 <u>and</u> y=0	
<b><u>APLIKASI</u></b>	
⇒ IsOrigin?(1,0) ⇒ IsOrigin? (1,1) ⇒ IsOrigin? (0,0)	

### Contoh-7 Ekspresi boolean dengan operator boolean: APAKAH VALID

#### Persoalan :

Buatlah definisi, spesifikasi dan realisasi dari sebuah predikat yang menerima sebuah besaran integer, dan menentukan apakah bilangan tersebut valid. Bilangan disebut valid jika nilainya lebih kecil dari 5 atau lebih besar dari 500. Jadi bilangan di antara 5 dan 500 tidak valid.

APAKAH VALID	IsValid?(x)
<b><u>DEFINISI DAN SPESIFIKASI</u></b>	
<b>IsValid?</b> : <u>integer</u> → <u>boolean</u> <i>{IsValid? (x) benar jika (x) bernilai lebih kecil 5 atau lebih besar dari 500 }</i>	
<b><u>REALISASI</u></b>	
<b>IsValid?</b> (x) : x <5 <u>or</u> x>500	
<b><u>APLIKASI</u></b>	
⇒ IsValid?(5) ⇒ IsValid? (0) ⇒ IsValid? (500) ⇒ IsValid? (6000)	

## Evaluasi Fungsi

### Evaluasi fungsi :

Suatu ekspresi dituliskan dan dihitung hasilnya (dievaluasi) sesuai dengan aturan pemroses bahasanya. Operator dapat dianggap sebagai fungsi yang paling dasar yang dipunyai oleh bahasa. Lihat penulisan dalam bentuk prefix:

\* 2 3

yang dapat kita pandang sebagai “fungsi” \*(2,3) dalam notasi fungsional.

Evaluasi ekspresi dalam konteks fungsional adalah melakukan aplikasi fungsi sambil melakukan evaluasi dari ekspresi yang mengandung operan. Karena sebuah ekspresi dapat mengandung lebih dari satu operan dan aplikasi fungsi, maka urutan dari evaluasi dapat bermacam-macam dan prioritas dari operan menentukan urutan evaluasi. Untuk ketepatan evaluasi dari ekspresi yang mengandung operan, disarankan untuk menuliskan tanda kurung secara eksplisit. Untuk evaluasi yang dilakukan berdasarkan aplikasi fungsi, kita harus mempelajari aturan evaluasi dari bahasa yang bersangkutan.

### Contoh-1 Evaluasi fungsi :

#### Persoalan :

Buatlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menerima empat buah bilangan riil yang pengertiannya adalah dua pasang titik pada koordinat kartesian, dan menghasilkan sebuah bilangan riil yang merupakan jarak dari kedua titik tersebut (atau panjang garis yang dibentuk oleh kedua titik tersebut), dengan melakukan aplikasi terhadap dua buah fungsi antara yang harus didefinisikan terlebih dulu sebagai berikut :  
**dif2** adalah sebuah fungsi yang menerima dua buah bilangan riil dan menghasilkan **pangkat dua dari selisih kedua bilangan riil** tersebut. Pangkat dua dilakukan oleh fungsi **quad** yang menerima sebuah bilangan riil dan menghasilkan **pangkat dua** dari bilangan riil tersebut

JARAK2TITIK, Least Square	LS(x1,x2,y1,y2)
<b><u>DEFINISI DAN SPESIFIKASI</u></b>	
LS : 4 <u>real</u> → <u>real</u> {LS(x1,x2,y1,y2) adalah jarak antara dua buah titik (x1,x2) dengan (y1,y2) }	
<b><u>DEFINISI DAN SPESIFIKASI FUNGSI ANTARA</u></b>	
dif2 : 2 <u>real</u> → <u>real</u> {dif(x,y) adalah kuadrat dari selisih antara x dan y }	
FX2 : <u>real</u> → <u>real</u> { FX2 (x) adalah hasil kuadrat dari x }	
<b><u>REALISASI</u></b>	
FX2 (x) : x * x dif2 (x,y) : FX2 (x - y) LS (x1,y1,x2,y2) : √ dif2 (y2,y1) + dif2 (x2,x1)	

Berikut ini diberikan contoh perhitungan yang dilakukan oleh “pemroses bahasa” pada saat aplikasi, yaitu jika evaluasi dilakukan dari kiri ke kanan.

**Evaluasi LS (1,3,5,6)**

```

--> V dif2(6,3) + dif2(5,1)      {pilih dif(6,3)
                                     untuk dievaluasi dulu }
--> V FX2(6-3) + dif2(5,1)      {ekspansi dif(6,3)}
--> V FX2(3) + dif2(5,1)      {reduksi, hasil evaluasi -}
--> V 3 * 3 + dif2(5,1)      {ekspansi quad(3)}
--> V 9 + dif2(5,1)          {reduksi *}
--> V 9 + FX2(5-1)          {ekspansi dif(5,1)}
--> V 9 - FX2(4)            {reduksi -}
--> V 9 + 4 * 4              {ekspansi quad(4)}
--> V 9 + 16                {reduksi *}
--> V 25                    {reduksi +}
--> 5                          {reduksi V }

```

Jika ekspresi mengandung aplikasi dari beberapa fungsi, secara teoritis beberapa evaluasi dapat dilakukan secara paralel karena evaluasi suatu fungsi asalakan parameternya siap dipakai akan dapat dilakukan secara independent terhadap evaluasi fungsi yang lain.

Perhatikan urutan evaluasi yang dilakukan di atas adalah berdasarkan “pilihan” dari pemroses. Untuk bahasa pemrograman fungsional yang nyata (misalnya LISP), perlu dipelajari urutan evaluasi yang dilakukan supaya didapatkan hasil yang sesuai dengan yang diinginkan. Urutan evaluasi seperti contoh di atas bahkan tidak mungkin diatur dengan menuliskan tanda kurung. Hal ini secara spesifik akan diuraikan pada Bagian kedua buku ini.

Selanjutnya, pada bagian ke satu, fokus kita adalah pada definisi dan realisasi fungsi yang hasilnya benar sesuai kaidah aritmatika. Untuk contoh di atas: operator + dalam aritmatika adalah operator yang komunitatif: a+b=b+a. Sebaiknya konstruksi dari program fungsional dibuat tidak tergantung (atau sesedikit mungkin tergantung) pada urutan evaluasi. Misalnya untuk suatu ekspresi aritmatika, jika mungkin, dituliskan di antara tanda kurung.

**Ekspresi Bernama, Nama Antara**

Suatu ekspresi “antara”, yaitu ekspresi yang dituliskan untuk sementara di dalam fungsi namun tidak dikomunikasikan ke dunia luar fungsi tersebut, dapat diberi nama (yang bukan merupakan nama fungsi). Nama ini hanya berlaku sementara di “dalam” sebuah fungsi.

Pemakaian fungsi atau nama antara tergantung pada generalitas solusi yang kita inginkan. Fungsi selalu dapat digunakan dalam konteks lain.

Type dari nama lokal yang dipakai menyimpan hasil ekspresi tidak perlu dinyatakan secara eksplisit. Type dapat dideduksi dari operator ekspresi.

Bentuk umum

```
<NAMA-FUNGSI> :  
    let <Nama> = <Ekspresi> in  
        <Realisasi-Fungsi>
```

dengan :

- *Nama-Fungsi* adalah Nama Fungsi yang direalisasi
- *Nama* adalah nama sementara bersifat “lokal” yang dipakai untuk menyimpan hasil evaluasi Ekspresi dan nilai hanya terdefinisi pada lingkup let di mana Fungsi tersebut direalisasi
- *Ekspresi* adalah suatu ekspresi fungsional
- *Realisasi-Fungsi* adalah Ekspresi Fungsional, realisasi dari Nama-Fungsi

Bentuk lebih Umum :

```
<NAMA-FUNGSI > :  
    let <Nama-1> = <Ekspresi-1>,  
        <Nama-2> = <Ekspresi-2>,  
        ... <Nama-k> = <Ekspresi-k> in  
        <Realisasi-Fungsi>
```

Perhatikan bahwa masing-masing <Ekspresi-I> akan dievaluasi independent dalam lingkup <NAMA-FUNGSI> dan bukan merupakan sekuens

Cara Evaluasi pada umumnya: untuk semua nilai *i*, evaluasi <Ekspresi-*i*>, dan gantikan semua kemunculan nama <Nama-*i*> dalam <NAMA-FUNGSI> dengan nilai *Ekspresi*. Nama adalah lokal terhadap *Fungsi*. Nilai-nilai dan ekspresi *Ekspresi* hanya ada artinya didalam *Fungsi*.

### **Pemakaian let**

Pemakaian **let ... in...** adalah untuk :

- menghindari evaluasi berulang-ulang.
- menjadikan program lebih mudah dibaca

### Menghindari evaluasi yang berulang

Misalnya dalam ekspresi  $(1+a*b) * (1-2*a*b)$  harus dilakukan evaluasi  $a * b$  sebanyak dua kali

Untuk menghindari evaluasi berulang dapat ditulis :

```
F(a,b) :  
  let p = a * b in  
    (1+p) * (1- 2*p)
```

### Memudahkan interpretasi/pembacaan teks program :

Misalnya pada MO (u,v,w,x)

#### REALISASI

```
MO (u,v,w,x) :  
  let S = u+v+w+x in  
    (S - min4(u,v,w,x) - max4(u,v,w,x)) / 2
```

#### REALISASI

```
MO (u,v,w,x) :  
  let S = u+v+w+x  
    M = max2 (max2 (max2 (u,v),w),x)  
    m = min2 (min2 (min2 (u,v),w),x)  
  in (S- m - M) / 2
```

Sebaiknya nama dalam huruf kecil dan huruf kapital semacam ini dihindarkan, karena membingungkan (beberapa bahasa bahkan menganggap sama dan menimbulkan konflik)

### Let yang mengandung Let

Suatu blok let dapat mengandung blok let di dalamnya, dituliskan sebagai berikut :

```
<Nama-Fungsi>  
  let Nama-1 = Ekspresi-1 in  
    let Nama-2 = Ekspresi-2 in  
      <Realisasi-Fungsi>
```

Dalam hal ini, Konteks/scope harus diperhatikan

Perhatikan Contoh berikut yang “membingungkan” :

```
F(x,y) :  
  let x = 3 + 4 * 5 in  
    let y = x + 5 in  
      x + y
```

Sebaiknya semua nama lokal tidak sama dengan nama parameter formal.

# 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 enumerasi 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 ekuivalen 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 ekuivalen 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 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 <u>AND then</u> B	<u>if</u> A <u>then</u> B <u>else</u> false

### OPERATOR OR else

B hanya dievaluasi jika Ekspresi A bernilai false.

Ekspresi boolean	Ekivalen dengan
A <u>OR else</u> B	<u>if</u> A <u>then true else</u> 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 ekspresi fungsional karena harus ada nilai untuk semua kasus.

Ekspresi kondisional yang menghasilkan nilai boolean sebagai berikut

<p><u>depend on</u> {deskripsi domain} &lt;Kondisi-1&gt; : &lt;Ekspresi-1&gt; &lt;Kondisi-2&gt; : &lt;Ekspresi-2&gt; &lt;Kondisi-3&gt; : &lt;Ekspresi-3&gt;</p>
---

dapat ditulis dalam bentuk lain yang ekivalen sebagai berikut.

<p><u>depend on</u> {deskripsi domain} &lt;Kondisi-1&gt; <u>AND then</u> &lt;Ekspresi-1&gt; <u>or</u> &lt;Kondisi-2&gt; <u>AND then</u> &lt;Ekspresi-2&gt; <u>or</u> &lt;Kondisi-3&gt; <u>AND then</u> &lt;Ekspresi-3&gt;</p>
---

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

<b>MAKSIMUM 2 NILAI</b>	<b>max2(a,b)</b>
<b><u>DEFINISI DAN SPESIFIKASI</u></b>	
<b>max2</b> : 2 <u>integer</u> → <u>integer</u> (max2 (a,b) menghasilkan maksimum dari 2 bilangan integer a dan b }	
<b><u>REALISASI</u></b>	
{ Notasi if then else sebab hanya ada dua kasus komplementer... } <b>max2</b> (a,b) : <u>if</u> a>b <u>then</u> a <u>else</u> 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.

<b>MAKSIMUM 3 NILAI</b>	<b>max3(a,b,c)</b>
<b><u>DEFINISI DAN SPESIFIKASI</u></b>	
<b>max3</b> : 3 <u>integer</u> → <u>integer</u> (max3(a,b,c)menentukan maksimum dari 3 bilangan integer yang berlainan nilainya, a ≠ b dan b ≠ c dan a ≠ c }	

Versi 1 : Identifikasi domain , berangkat dari hasil :

<b>MAKSIMUM 3 NILAI (versi 1)</b>	<b>max3(a,b,c)</b>
<b><u>DEFINISI DAN SPESIFIKASI</u></b>	
<p><b>max3</b> : 3 <u>integer</u> → <u>integer</u>  <i>(max3(a,b,c)menentukan maksimum dari 3 bilangan integer yang berlainan nilainya, a ≠ b dan b ≠ c dan a ≠ c }</i></p>	
<b><u>REALISASI</u></b>	
<p><b>max3</b> (a, b, c) :              <u>depend on</u> a, b, c                  a &gt; b <u>and</u> a &gt; c: a                  b &gt; a <u>and</u> b &gt; c: b                  c &gt; a <u>and</u> c &gt; b: c</p>	

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.

<b>MAKSIMUM 3 NILAI (versi 2)</b>	<b>max3(a,b,c)</b>
<b><u>DEFINISI DAN SPESIFIKASI</u></b>	
<p><b>max3</b> : 3 <u>integer</u> → <u>integer</u>  <i>(max3(a,b,c)menentukan maksimum dari 3 bilangan integer yang berlainan nilainya, a ≠ b dan b ≠ c dan a ≠ c }</i></p>	
<b><u>REALISASI</u></b>	
<p><b>max3</b> (a, b, c) :              <u>depend on</u> a, b, c                  a &gt; b <u>and</u> b &gt; c: a                  a &gt; c <u>and</u> c &gt; b: a                  b &gt; a <u>and</u> a &gt; c: b                  b &gt; c <u>and</u> c &gt; a: b                  c &gt; a <u>and</u> a &gt; b: c                  c &gt; b <u>and</u> b &gt; a: c</p>	

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.

<b>MAKSIMUM 3 NILAI (versi 3)</b>	<b>max3(a,b,c)</b>
<b><u>DEFINISI DAN SPESIFIKASI</u></b>	
<p><b>max3</b> : 3 <u>integer</u> → <u>integer</u></p>	

*(max3(a,b,c)menentukan maksimum dari 3 bilangan integer yang berlainan nilainya, a ≠ b dan b ≠ c dan a ≠ 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

<b>MAKSIMUM 3 NILAI (versi 4)</b>	<b>max3(a,b,c)</b>
<b><u>DEFINISI DAN SPESIFIKASI</u></b>	
<p><b>max3</b> : tiga <u>integer</u> → <u>integer</u></p> <p><i>(max3(a,b,c)menentukan maksimum dari 3 bilangan integer yang berlainan nilainya, a ≠ b dan b ≠ c dan a ≠ c }</i></p>	
<b><u>REALISASI</u></b>	
<pre> <b>max3</b> (a,b,c)   <u>let</u> m = <u>depend on</u> a,b                                 a&gt;b:a                                 a&lt;b:b   <u>in</u>   <u>depend on</u> m,c                 m&gt;c:m                 m&lt;c:c </pre>	

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

<b>MAKSIMUM 3 NILAI (versi 5)</b>	<b>max3 (a,b,c)</b>
<b><u>DEFINISI DAN SPESIFIKASI</u></b>	
<p><b>max3</b> : 3 <u>integer</u> → <u>integer</u></p> <p><i>(max3(a,b,c)menentukan maksimum dari 3 bilangan integer yang berlainan nilainya, a ≠ b dan b ≠ c dan a ≠ c }</i></p>	
<b><u>REALISASI</u></b>	
<p><b>max3</b> (a,b,c) : max2 (max2 (a,b, ) , c)</p>	

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

<b>MAKSIMUM 3 NILAI (versi 6)</b>	<b>max3 (a,b,c)</b>
<b>DEFINISI DAN SPESIFIKASI</b>	
<p><b>max3</b> : 3 <u>integer</u> → <u>integer</u>  <i>(max3(a,b,c)menentukan maksimum dari 3 bilangan integer yang berlainan nilainya, a ≠ b dan b ≠ c dan a ≠ c }</i></p>	
<b>REALISASI</b>	
<b>max3</b> (a, b, c) : max2 (c, max2 (a, b))	
<b>MAKSIMUM 3 NILAI (versi 7)</b>	<b>max3 (a,b,c)</b>
<b>DEFINISI DAN SPESIFIKASI</b>	
<p><b>max3</b> : 3 <u>integer</u> → <u>integer</u>  <i>(max3(a,b,c)menentukan maksimum dari 3 bilangan integer yang berlainan nilainya, a ≠ b dan b ≠ c dan a ≠ c }</i></p>	
<b>REALISASI</b>	
<b>max3</b> (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

<b>PENANGGALAN</b>	<b>HariKe1900(d,m,y)</b>
<b>DEFINISI DAN SPESIFIKASI</b>	
<p><b>Harike1900</b> : <u>integer</u> [1..31], <u>integer</u> [1..12] <u>integer</u> [0..99] → <u>integer</u> [1..366]  <i>{Harike1900(d,m,y) dari suatu tanggal &lt;d,m,y&gt; adalah hari 'absolut' dihitung mulai 1 Januari 1900+y. 1 Januari tahun 1900+y adalah hari ke 1}</i></p> <p><b>dpm</b> : <u>integer</u> [1..12] → <u>integer</u> [1..36]  <i>{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}</i></p>	

### 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  $\langle j,m,s \rangle$  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  $\langle 2,20,15 \rangle$  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**  $\langle x:\underline{\text{integer}}, y:\underline{\text{integer}} \rangle$ , dipakai notasi :  $\langle 0,0 \rangle$  sebagai Titik Origin,  $\langle 1,2 \rangle$  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  $\langle \text{absis}, \text{ordinat} \rangle$  bertype  $\langle \text{integer}, \text{integer} \rangle$
2. Type Bilangan Kompleks, yang terdiri dari bagian riil dan bagian imajiner: yang bertype bilangan riil  $\langle \text{riil}, \text{imaginer} \rangle$
3. Type Pecahan, yang terdiri dari  $\langle \text{pembilang}, \text{penyebut} \rangle$  yang bernilai  $\langle \text{integer}, \text{integer} \rangle$
4. Type JAM, yang terdiri dari  $\langle \text{jam}, \text{menit}, \text{detik} \rangle$
5. Type DATE yang terdiri dari  $\langle \text{tanggal}, \text{bulan}, \text{tahun} \rangle$

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 ketersediaan 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.

<b>TYPE POINT</b>
<b><u>DEFINISI TYPE</u></b> <b>type point</b> : $\langle x: \text{real}, y: \text{real} \rangle$ <i>{<math>\langle x,y \rangle</math> adalah sebuah point, dengan <math>x</math> adalah absis, <math>y</math> adalah ordinat }</i>
<b><u>DEFINISI DAN SPESIFIKASI SELEKTOR</u></b> <b>Absis</b> : point $\rightarrow$ <u>real</u> <i>{Absis(P) Memberikan Absis Point P}</i> <b>Ordinat</b> : point $\rightarrow$ <u>real</u> <i>{Ordinat(P) Memberikan ordinat Point P }</i>
<b><u>DEFINISI DAN SPESIFIKASI KONSTRUKTOR</u></b> <b>MakePoint</b> : 2 <u>real</u> $\rightarrow$ point <i>{ MakePoint(a,b) membentuk sebuah point dari a dan b dengan a sebagai absis dan b sebagai ordinat }</i>
<b><u>DEFINISI DAN SPESIFIKASI PREDIKAT</u></b> <b>IsOrigin?</b> : point $\rightarrow$ <u>boolean</u> <i>{ IsOrigin?(P) benar jika P adalah titik origin yaitu titik <math>\langle 0,0 \rangle</math> }</i>
<b><u>DEFINISI OPERATOR/FUNGSI LAIN TERHADAP POINT</u></b> <b>Jarak</b> : 2 point $\rightarrow$ <u>real</u> <i>{Jarak(P1,P2) : menghitung jarak antara 2 point P1 dan P2 }</i> <b>Jarak0</b> : point $\rightarrow$ <u>integer</u> <i>{ Jarak0(P1) Menghitung jarak titik terhadap titik pusat koordinat (0,0) }</i> <b>Kuadran</b> : point $\rightarrow$ <u>integer</u> [1..4] <i>{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}</i> <i>{ Fungsi antara yang dipakai : FX2 adalah pangkat dua yang pernah didefinisikan pada least square dan SQRT(X) adalah fungsi dasar untuk menghitung akar}</i>

## 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 (denominator). 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 denominator 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 samainilainya:  $n1/d1 = n2/d2$  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

```
IsBefore? (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 bertipe 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 tipe 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)
<b><u>DEFINISI DAN SPESIFIKASI</u></b>	
<p><b>HHMMDD (x)</b> : <math>\text{integer } [0..99999] \rightarrow \langle \text{integer} &gt; 0, \text{integer}[0..23], 2 \text{ integer } [0..59] \rangle</math>  <i>{ mis. <math>\langle a,b,c,d \rangle = \text{HHMMDD}(x), x = 86400 * a + 3600 * b + 60 * c + d \}</math></i></p> <p><b>QR</b> : <math>\langle \text{integer} &gt; 0, \text{integer} &gt; 0 \rangle \rightarrow \langle \text{integer} &gt; 0, \text{integer} &gt; 0 \rangle</math>  <i>{let <math>\langle q,r \rangle = \text{QR} \langle N,D \rangle</math> : <math>q</math> adalah hasil bagi bulat  <math>r</math> adalah sisa pembagian bulat dari <math>N</math> dibagi <math>D</math>  <math>N = q * D + r</math> and <math>0 &lt; r, r &lt; D\}</math></i></p>	
<b><u>REALISASI</u></b>	
<p><b>QR</b> (<math>\langle N,D \rangle</math>) : <math>\langle N \text{ div } D, N \text{ mod } D \rangle</math></p> <p><b>HHMMDD</b> (x) :</p> <pre> let &lt; H, sisaH &gt; = QR (x, 86400) in   let &lt; J, sisaJ &gt; = QR (sisaH, 3600) in     let &lt; M, D &gt; = QR (sisa J, 60) in       &lt; H, J, M, D &gt; </pre>	
<b><u>REALISASI –VERSI TANPA LET</u></b>	
<p><b>QR</b> (<math>\langle N,D \rangle</math>) : <math>\langle N \text{ div } D, N \text{ mod } D \rangle</math></p> <p><b>HHMMDD</b> (x) :</p> <pre> &lt; x div 86400, (x mod 8600) div 3600,   ((x mod 8600) div 3600) div 60, ((x mod 8600) div 3600) mod 60 &gt; </pre>	

Perhatikanlah bahwa notasi  $\langle a,b \rangle$  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 ditranslasikan 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 bertipe 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 <i>domain</i> dan <i>range</i>
Konstruktor untuk Membuat koleksi kosong	→ Koleksi
Selektor, untuk melakukan akses terhadap elemen koleksi objek	Koleksi → elemen
Predikat untuk melakukan test : <ul style="list-style-type: none"> <li>▪ apakah koleksi kosong</li> <li>▪ apakah koleksi masih mampu menampung objek baru (belum penuh)</li> </ul>	Koleksi → <u>boolean</u>
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 ( <i>search, membership</i> )	Koleksi, elemen → <u>boolean</u>

Operasi	Definisi <i>domain</i> dan <i>range</i>
Predikat yang merupakan operator relasional untuk membandingkan dua buah koleksi objek	Koleksi, Koleksi → <u>boolean</u>

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, kumpulan elemen yang diorganisasi 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  $[b_i \dots b_s]$  dan memetakan pada harga tertentu.  $b_i \leq b_s$

$b_i$  = 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  $T$

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

## Perbedaan tabel [...] dengan type komposisi

- elemen tabel bertipe sama,
- elemen type komposisi boleh bermacam-macam
- elemen type komposisi telah diketahui dengan pasti dan fixed pada pendefinisinya, 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  $\langle d,m,y \rangle$  dan menghasilkan Harike pada tanggal tersebut, terhitung mulai 1 Januari tahun yang bersangkutan. Contoh masukan dan hasil :

$\langle 1,1,1993 \rangle \rightarrow 1$

$\langle 4,2,1979 \rangle \rightarrow 35$

$\langle 31,12,1935 \rangle \rightarrow 365$

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):

<b>0</b>	<b>1</b>	<b>-1</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>3</b>	<b>4</b>	<b>4</b>
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 :

<b>BULAN</b>	<b>JUMLAH HARI</b>	<b>KOREKSI</b>
1	1	0
2	32	$1 + 1 \times 30 + 1$
3	60	$1 + 2 \times 30 + (-1)$
4	91	$1 + 3 \times 30 + 0$
5	121	$1 + 4 \times 30 + 0$
6	152	$1 + 5 \times 30 + 1$
7	182	$1 + 6 \times 30 + 1$
8	213	$1 + 7 \times 30 + 2$
9	244	$1 + 8 \times 30 + 3$
10	274	$1 + 9 \times 30 + 3$
11	305	$1 + 10 \times 30 + 4$
12	335	$1 + 11 \times 30 + 4$

HARIKE (versi-0)	NH(d,m,y)
<b><u>DEFINISI DAN SPESIFIKASI TYPE</u></b>	
<p><b>type Hr</b> : <u>integer</u> [1...31]</p> <p>{definisi ini hanyalah untuk memberi nama baru terhadap type integer dengan nilai tertentu supaya mewakili hari, sehingga jika dipunyai suatu nilai integere, kita dapat memeriksa apakah nilai integer tersebut mewakili Hari yang absah }</p> <p><b>type Bln</b> : <u>integer</u> [1...12]</p> <p>{definisi ini hanyalah untuk memberi nama baru terhadap type integer dengan nilai tertentu supaya mewakili Bulan }</p> <p><b>type Thn</b> : <u>integer</u> &gt; 0</p> <p>{definisi ini hanyalah untuk untuk memberi nama baru terhadap type integer dengan nilai tertentu supaya mewakili tahun }</p> <p><b>type date</b> &lt;d: Hr, m:Bln,y:Thn&gt;</p> <p>{ &lt;d,m,y&gt; adalah tanggal d bulan m tahun y }</p> <p><b>type NHari</b> : <u>integer</u> [1..366]</p> <p>{Jumlah hari dalam suatu tahun kalender }</p>	
<b><u>DEFINISI DAN SPESIFIKASI SELEKTOR DENGAN FUNGSI</u></b>	
<p><b>Day</b> : date → Hr</p> <p>{Day (D) memberikan hari d dari D yang terdiri dari &lt;d,m,y&gt; }</p> <p><b>Month</b> : date → Bln</p> <p>{Month (D) memberikan bulan m dari D yang terdiri dari &lt;d,m,y&gt; }</p> <p><b>Year</b> : date → Thn</p> <p>{Year (D) memberikan tahun y dari D yang terdiri dari &lt;d,m,y&gt; }</p>	
<b><u>DEFINISI TABEL KOREKSI</u></b>	
Koreksi : <u>table of</u> Bln = [0,1,-1,0,0,1,1,2,3,3,4,4]	
<b><u>DEFINISI DAN SPESIFIKASI FUNGSI ANTARA</u></b>	
<p><b>IsKabisat?</b> : <u>integer</u> → <u>boolean</u></p> <p>{IsKabisat?(a) true jika tahun 1900+a adalah tahun kabisat: habis dibagi 4 tetapi tidak habis dibagi 100, atau habis dibagi 400 }</p>	
<b><u>DEFINISI DAN SPESIFIKASI FUNGSI NH</u></b>	
<p><b>NH</b> : date →Nhari</p> <p>{NH(&lt;d,m,y&gt; menghasilkan hari absolut pada yahun y; setiap bln bervariasi sesuai dengan jumlah hari dalam bulan ybs: Januari : 30 hari; Februari 28 atau 29 hari; maret : 31 hari,.... Jadi realisasi versi-0 harus dikoreksi dengan tabel koreksi}</p>	
<b><u>REALISASI KOREKSI DENGAN TABEL</u></b>	
<p><b>NH</b> (D) :     30 * (Month (D)-1) + Day (D) + Koreksi<sub>B</sub> +                    <u>if</u> m&gt;2 <u>and</u> IsKabisat? Year (D) <u>then</u> 1 <u>else</u> 0</p> <p><b>IsKabisat?</b>(a) : ( (a mod 4 = 0) <u>and</u> (a mod 100 ≠ 0) ) <u>or</u> (a <u>div</u> 400 =0)</p>	

**Contoh 2:**

Pada contoh berikut, realisasi koreksi akan dituliskan sebagai fungsi dan bukan sebagai tabel. Realisasinya menjadi suatu ekspresi kondisional yang pernah dibahas. Untuk kasus ini, karena dalam satu tahun “hanya” ada 12 bulan, dan untuk nilai koreksi hanya dibutuhkan satu nilai, maka hanya dibutuhkan sebuah nama dan sebuah analisa kasus terhadap 12 kemungkinan harga yang harus dihasilkan oleh fungsi.. **Solusi ini tidak memakai type berupa koleksi objek**, berbeda halnya dengan solusi dengan tabel.

<b>HARIKE (versi-Fungsi)</b>	<b>NH(d,m,y)</b>
<p><b><u>DEFINISI DAN SPESIFIKASI TYPE</u></b></p> <p><b>type Hr</b> : <u>integer</u> [1...31]  <i>{definisi ini hanyalah untuk “menamakan kembali 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 }</i></p> <p><b>type Bln</b> : <u>integer</u> [1...12]  <i>{definisi ini hanyalah untuk “menamakan kembali type integer dengan nilai tertentu supaya mewakili Bulan }</i></p> <p><b>type Thn</b> : <u>integer</u> &gt; 0  <i>{definisi ini hanyalah untuk “menamakan kembali type integer dengan nilai tertentu supaya mewakili tahun }</i></p> <p><b>type date</b> &lt;d: Hr, m:Bln,y:Thn&gt;  <i>{ &lt;d,m,y&gt; adalah tanggal d bulan m tahun y }</i></p> <p><b>type NHari</b> : <u>integer</u> [1..366]  <i>{Jumlah hari dalam suatu tahun kalender }</i></p>	
<p><b><u>DEFINISI DAN SPESIFIKASI SELEKTOR DENGAN FUNGSI</u></b></p> <p><b>Day</b> : date → Hr  <i>{Day (D) memberikan hari d dari D yang terdiri dari &lt;d,m,y&gt; }</i></p> <p><b>Month</b> : date → Bln  <i>{Month (D) memberikan bulan m dari D yang terdiri dari &lt;d,m,y&gt; }</i></p> <p><b>Year</b> : date → Thn  <i>{Year (D) memberikan tahun y dari D yang terdiri dari &lt;d,m,y&gt; }</i></p>	
<p><b><u>DEFINISI DAN SPESIFIKASI FUNGSI NH</u></b></p> <p><b>NH</b> : date → NHari  <i>{NH(&lt;d,m,y&gt; menghasilkan hari absolut pada tahun y; setiap bln bervariasi sesuai dengan jumlah hari dalam bulan ybs: Januari : 30 hari; Februari 28 atau 29 hari; maret : 31 hari,... }</i></p>	

## DEFINISI DAN SPESIFIKASI FUNGSI ANTARA

**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 }*

**Koreksi(B)** : Bln →integer

*{Koreksi (B) adalah banyaknya koreksi terhadap perhitungan hari s/d tanggal 1 bulan B, }*

## REALISASI

**NH(D)** :  $30 * (\text{Month (D)} - 1) + \text{Day (D)} + \text{Koreksi}(\text{Month (D)}) +$   
 $\text{if } m > 2 \text{ and IsKabisat? Year (D) then } 1 \text{ else } 0$

**IsKabisat?(a)** :  $( (a \bmod 4 = 0) \text{ and } (a \bmod 100 \neq 0) ) \text{ or } (a \bmod 400 = 0)$

## REALISASI KOREKSI DENGAN FUNGSI

{ Pada solusi ini, fungsi Koreksi(B) menggantikan tabel Koreksi pada solusi sebelumnya }

Koreksi(B) : depend on B  
B = 1 : 0  
B = 2 : 1  
B = 3 : -1  
B = 4 : 0  
B = 5 : 0  
B = 6 : 1  
B = 7 : 1  
B = 8 : 1  
B = 9 : 2  
B = 10 : 3  
B = 11 : 3  
B = 12 : 4

## EKSPRESI REKURSIF

Definisi entitas (type, fungsi) disebut rekursif jika definisi tersebut mengandung terminologi dirinya sendiri.

Ekspresi rekursif dalam pemrograman fungsional didasari oleh Analisa rekurens, yaitu penalaran berdasarkan definisi **fungsi rekursif**, yang biasanya juga berdasarkan “type” yang juga terdefinisi secara rekursif-induktif.

Bandingkanlah cara induksi ini dengan pendekatan eksperimental klasik, yang mencoba-coba menurunkan kebenaran dari observasi, atau dapat juga dengan cara berusaha menemukan *counter-example* (contoh yang salah, yang menunjukkan kebalikannya).

Contoh cara pembuktian kebenaran dengan menemukan contoh yang salah :

$$f(n) = n^2 - n + 41$$

$$f(0) = 41 \quad f(1) = 41 \quad f(2) = 43$$

$$f(3) = 47 \quad f(4) = 53$$

$f(n)$  adalah fungsi untuk menghasilkan bilangan prima : benar untuk  $n = 40$ , tetapi untuk  $n = 41$  salah, karena  $41^2 - 41 + 41 = 1681$  bukan bilangan prima.

Metoda pembuktian rekurens : metoda dimana sifat (*property*) dibuktikan benar untuk satu elemen (sebagai basis), kemudian benar untuk semua elemen

### Bukti secara rekurens

Bukti rekurens adalah Cara untuk membuktikan bahwa *property* (suatu sifat) adalah benar untuk semua elemen:

- dengan basis benar jika  $n=0$ ;
- kemudian untuk  $n$  dianggap benar, kita membuktikan untuk  $n+1$  benar sehingga dapat menyimpulkan bahwa untuk semua  $n$  benar.

#### a. **Bukti rekurens terhadap bilangan integer.**

- Basis : property untuk  $n = 0$

- Rekurens : benar untuk  $n$   
benar untuk  $n+1$

#### **Contoh -pembuktian-1**

Buktikan bahwa  $10^n - 1$  habis dibagi 9.

Bukti : A (bilangan natural) habis dibagi 9 jika ada  $n$  sehingga  $A=n*9$

Basis : untuk  $n= 0$  benar, sebab  $10^n - 1 = 0$  habis dibagi 9

Rekurens : untuk  $n$  sembarang, dianggap bahwa  $10^n - 1$  habis dibagi 9.

$$10^{n+1} - 1 = 10 * (10^n - 1) + 9$$

$$10 (10^n) - 1 = 9 * (10 * 9)$$

$$10 * (10^n - 1) + 10 - 1$$

### Contoh-pembuktian -2

Buktikan bahwa semua bilangan bulat  $\geq 2$  dapat diuraikan dalam faktor primer, yaitu dapat diekspresikan sebagai hasil kali dari faktor primer.

Basis : benar untuk  $n=2$  dengan analisa kasus :

Rekurens:

- jika  $n$  bilangan prima, maka  $n$  dapat diuraikan dalam faktor primer (benar) sebab semua bilangan prima dapat difaktorisasi menjadi bilangan prima
- jika  $n$  bukan bilangan prima :  
Misalnya  $k$  dapat ditulis dalam faktor bilangan prima.  
 $k+1$  juga dapat ditulis dalam faktor bilangan prima,  $k+1 = d$   
ada  $2 \leq c, d \leq k$ . Karena  $2, 3, \dots, k$  mempunyai pembagi prima,  
maka  $c$  dan  $d$  dapat ditulis dengan faktor bilangan prima

### Contoh- Pembuktian -3

Buktikan bahwa banyaknya himpunan bagian dari suatu himpunan dengan kardinalitas  $n$  adalah  $2^n$

Basis : untuk himpunan kosong :  $2^0 = 1$

Sebuah himpunan adalah merupakan himpunan bagian dari diri sendiri.

Himpunan bagian dari suatu himpunan dengan kardinalitas  $n$  adalah  $2^n$ .

$E$  adalah himpunan dengan kardinalitas  $n+1$

Isolasi sebuah elemen  $a$  dari  $E$ , dan himpunan selain  $a$  disebut  $F$ .

Maka

$E - P$  himpunan. bagian (hanya mengandung  $a$ )

$P^1$  himpunan bagian (tanpa  $a$ )

$P$  dan  $P^1$  adalah *disjoint*

Misalnya kardinalitas  $P$  adalah  $n$

$P^1 : 2^n$

Kardinalitas  $P$  sama dengan  $P^1$  dengan menambahkan elemen  $a$  menjadi elemen  $P^1$  adalah  $2 * 2^n = 2^{n+1}$ .

### Type Rekurstif

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

Perhatikanlah beberapa contoh definisi type sebagai berikut:

#### a. Bilangan integer ganjil

Basis : 1 adalah bilangan integer ganjil

Rekurens : if  $x$  adalah bilangan integer ganjil

then  $x + 2$  adalah bilangan integer ganjil

**b. Luas bujur sangkar**

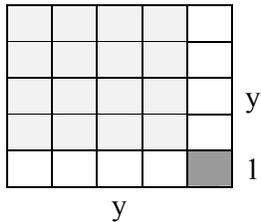
Basis : luas bujur sangkar dengan sisi 1 = 1

Rekurens :

Jika C adalah bujur sangkar dengan sisi y,

bujur sangkar dengan sisi y+1 didapat dengan menambahkan 1 pada setiap sisinya = y+1. Luas bujur sangkar menjadi :

$$(y+1)^2 = y^2 + 2Y + 1$$



**c. List/sequence :**

Basis : list Kosong : [ ] list tanpa elemen adalah sebuah list

Rekurens : list tidak kosong dibentuk dengan konstruktor

List **dibentuk** dengan menambahkan sebuah elemen menjadi anggota list

Konso(S,e) : S o e tambah sebuah element di 'kanan'

Konso•(e,S) : e•S tambah sebuah element di 'kiri'

**d. Bilangan natural :**

Basis : 0 adalah bilangan natural

Rekurens :

Jika x adalah bilangan natural,

Bilangan natural yang lain dibentuk dengan menambahkan 1 pada x, succ(x)

atau dengan mengalikan suatu bilangan natural dengan 2:  $x \rightarrow 2x$ ,

$d(x,0) \rightarrow$  untuk integer genap ( $2n$ )

$d(x,1) \rightarrow$  untuk integer ganjil ( $2n+1$ )

## Fungsi Rekursif

Dalam ekspresi fungsional: realisasi fungsi rekursif. Ada dua kategori fungsi rekursif, yaitu rekursif langsung atau tidak langsung.

### Fungsi Rekursif langsung

Fungsi didefinisikan rekursif, jika **ekspresi** yang merealisasi fungsi tersebut mengandung **aplikasi** terhadap fungsi tersebut. :

#### REALISASI

```
F (<list-param>) :  
  depend on  
    <kondisi-basis >      : <ekspresi-1 >  
    <kondisi-rekurens > : F (<ekspresi-2 >)
```

Dengan catatan, bahwa ekspresi-2 biasanya dinyatakan dengan domain yang sama dengan <list-param>, namun “mendekati” kondisi basis sehingga suatu saat akan terjadi kondisi basis yang menyebabkan aplikasi berhenti.

### Fungsi rekursif tidak langsung

Realisasi fungsi mungkin *cross-recursive*, yaitu jika realisasi fungsi F mengandung aplikasi fungsi G yang realisasinya adalah aplikasi terhadap F.

#### REALISASI

```
G (<list-param>): F (<ekspresi-1 >)  
  
F (<list-param>):  
  depend on  
    <kondisi-basis > : <ekspresi-1 >  
    <kondisi-rekurens > : G (<ekspresi-2 >)
```

Dalam menuliskan suatu fungsi rekursif, pemrogram harus membaca ulang teksnya, dan dapat “membuktikan” bahwa suatu saat program akan “berhenti”, karena mempunyai basis. Pembuktian secara matematis diluar cakupan diktat kuliah ini

Berikut ini akan diberikan contoh fungsi rekursif sederhana dan aplikasinya. Disebut “sederhana”, karena program rekursif benar-benar dibangun dari definisi rekursif dari persoalan yang akan dikomputasi, seperti definisi Faktorial dan Fibonacci.

## Contoh-1 Ekspresi rekursif : FACTORIAL

### Persoalan :

Tuliskanlah sebuah fungsi yang menghitung factorial dari  $n$  sesuai dengan **definisi rekursif** faktorial.

Faktorial	fac(n)
<b><u>DEFINISI DAN SPESIFIKASI</u></b>	
$\text{fac} : \text{integer} \geq 0 \rightarrow \text{integer} > 0$ { <i>fac (n) = n!</i> sesuai dengan definisi rekursif factorial }	
<b><u>REALISASI (VERSI-1)</u></b>	
{ <i>Realisasi dengan definisi factorial sebagai berikut jika fac(n) adalah n!</i> : $n = 0 : n! = 1$ $n \geq 1 : n! = (n-1)! * n$ }	
$\text{fac (n) : } \underline{\text{if}} \ n = 1 \ \underline{\text{then}} \ \{\text{Basis 1}\}$ 1 $\underline{\text{else}} \ \{\text{Rekurens : definisi faktorial}\}$ fac (n-1) * n	
<b><u>REALISASI (VERSI-2)</u></b>	
{ <i>Realisasi dengan definisi factorial sebagai berikut jika fac(n) adalah n!</i> : $n = 0 : n! = 1$ $n \geq 1 : n! = (n-1)! * n$ }	
$\text{fac (n) : } \underline{\text{if}} \ n = 0 \ \underline{\text{then}} \ \{\text{Basis 0}\}$ 1 $\underline{\text{else}} \ \{\text{Rekurens : definisi faktorial}\}$ n * fac (n-1)	
<b><u>REALISASI (VERSI-3): RUMUS BENAR, PROGRAM YANG SALAH !</u></b>	
{ <i>Realisasi dengan definisi factorial sebagai berikut jika fac(n) adalah n!</i> : $n = 1 : n! = 1$ $n > 1 : n! = (n+1)! / n$ }	
{ <i>Realisasi berikut yang didasari definisi di atas tidak benar sebab evaluasi untuk fac(n), n&gt;1 menimbulkan pemanggilan yang tidak pernah berhenti</i> }	
$\text{fac (n) : } \underline{\text{if}} \ (n = 1) \ \underline{\text{then}} \ \{\text{Basis 1}\}$ 1 $\underline{\text{else}} \ \{\text{Rekurens : definisi faktorial}\}$ fac(n+1) / (n)	

### Contoh-3 Ekspresi rekursif : FIBONACCI

#### Persoalan :

Tuliskanlah sebuah fungsi yang menghitung Fibonacci dari  $n$  , dengan **definisi rekursif** fungsi Fibonacci.:

Fibonacci	Fib(n)
<p><b><u>DEFINISI DAN SPESIFIKASI</u></b></p> <p><b>Fib</b> : <math>\text{integer} \geq 0 \rightarrow \text{integer} \geq 0</math> { Definisi rekursif fungsi fibonacci : } { Fib (n) = sesuai dengan definisi deret fibonacci :     <math>n = 0 : \text{Fib}(0) = 0</math>     <math>n = 1 : \text{Fib}(1) = 1</math>     <math>n &gt; 1 : \text{Fib}(n) = \text{Fib}(n-1) + \text{Fib}(n-2)</math> }</p>	
<p><b><u>REALISASI</u></b></p> <p><b>Fib</b> (n) : <u>depend on</u> (n)     <math>n = 0 : 0</math> {Basis 0}     <math>n = 1 : 1</math> {Basis 1}     <math>n &gt; 1 : \text{Fib}(n-1) + \text{Fib}(n - 2)</math> {Rekurens}</p>	

Perhatikanlah bahwa “basis” dari fungsi ini sesuai dengan dua buah aturan, yaitu untuk  $n=0$  dan  $n=1$ , karena rekurens dimulai dari  $n=2$

## Ekspresi Rekursif terhadap Bilangan Bulat

Pada bagian ini akan diberikan contoh, bagaimana menuliskan ekspresi rekursif yang dibangun berdasarkan analisa rekurens, terhadap bilangan integer, untuk merealisasi penjumlahan, perkalian dan pemangkatan bilangan integer.

Contoh yang dibahas dalam sub bab ini hanyalah memberikan pola berpikir rekursif terhadap type sederhana (integer). Dalam kenyataannya, memang pemrogram tidak menuliskan lagi fungsi rekursif untuk penjumlahan, pengurangan, pembagian karena sudah dianggap operator dasar.

Untuk menulis ekspresi rekursif dengan fungsi rekursif untuk bilangan integer, bilangan integer harus didefinisikan secara rekursif sebagai berikut:

- Basis : Nol adalah bilangan integer
- Rekurens :
  - Jika  $n$  adalah bilangan integer, maka suksesor dari  $n$  adalah bilangan integer
  - Jika  $n$  adalah bilangan integer, maka predesesor dari  $n$  adalah bilangan integer

TYPE INTEGER
<p><b>DEFINISI KONSTRUKTOR</b></p> <p>{ <b>Konstruktor <u>integer</u> <math>&gt; 0</math></b> }</p> <p><b>succ</b> : <u>integer</u> <math>\geq 0 \rightarrow</math> <u>integer</u> <math>&gt; 0</math></p> <p>{ Basis: <math>n = 0 \rightarrow 0</math> Rekurens : <math>n \rightarrow n + 1</math> <i>succ(n) membentuk deret bilangan <u>integer</u> positif}</i></p> <p>{ <b>Konstruktor <u>integer</u> <math>&lt; 0</math></b> }</p> <p><b>prec</b> : <u>integer</u> <math>\rightarrow</math> <u>integer</u></p> <p>{ Basis: <math>n =</math> representasi maksimal atau minimal mesin . Untuk <math>n</math> bilangan positif, basisnya seringkali diambil 0} Rekurens : <math>n \rightarrow n - 1</math> <i>prec(n) membentuk deret bilangan <u>integer</u> negatif }</i></p>
<p><b>REALISASI</b></p> <p><b>succ</b> (n) : <math>n + 1</math></p> <p><b>prec</b> (n) : <math>n - 1</math></p>

Berikut ini akan diberikan beberapa contoh fungsi rekursif berdasarkan definisi rekursif bilangan integer tersebut, dengan “memperkecil” menuju basisnya

### Contoh-1 Penjumlahan bilangan bulat dengan ekspresi rekursif

#### Persoalan :

Tuliskanlah sebuah fungsi yang menjumlahkan dua buah integer, dan menghasilkan sebuah integer, dengan membuat definisi rekursif dari penjumlahan

PENJUMLAHAN dua bilangan integer	Plus(x,y)
<b><u>DEFINISI DAN SPESIFIKASI</u></b>	
<b>Plus</b> : 2 <u>integer</u> > 0 → <u>integer</u> > 0 { Plus(x,y) menghasilkan x + y } }	
<b><u>REALISASI VERSI-1 : REKURENS TERHADAP Y</u></b>	
$\{ Plus(x,y) = x + y = x + 1 + 1 + \dots + 1$ $= x + 1 + (y-1) = 1 + x + (y-1) \}$ $\{ Basis : y = 0 \rightarrow x$ $Rekurens : y > 0 \rightarrow 1 + Plus(x, prec(y)) \}$ <b>Plus</b> (x,y) : <u>if</u> y = 0 <u>then</u> {Basis 0} x <u>else</u> {Rekurens terhadap y } 1 + Plus(x,prec(y))	
<b><u>REALISASI VERSI-2 : REKURENS TERHADAP X</u></b>	
$\{ Plus(x,y) = x + y = 1 + 1 + \dots + 1 + y$ $= 1 + (x-1) + y \}$ $\{ Basis : x = 0 \rightarrow y$ $Rekurens : x > 0 \rightarrow 1 + Plus(prec(x),y) \}$ <b>Plus</b> (x,y) : <u>if</u> x = 0 <u>then</u> {Basis 0} y <u>else</u> {Rekurens terhadap x} Plus(prec(x), y)	

## Contoh-2 Perkalian bilangan bulat dengan ekspresi rekursif

### Persoalan :

Tuliskanlah definisi, spesifikasi dan realisasi sebuah fungsi yang mengalikan dua buah integer, dan menghasilkan sebuah integer, dengan membuat definisi rekursif dari perkalian.

PERKALIAN dua bilangan integer	Mul(x,y)
<b>DEFINISI DAN SPESIFIKASI</b>	
<b>Mul</b> : 2 <u>integer</u> >0 → <u>integer</u> > 0 { Mul (x,y) menghasilkan $x * y$ }	
<b>REALISASI VERSI-1 : REKURENS TERHADAP Y</b>	
$\{ \text{Mul}(x,y) = x * y$ $= x + x + x + x \dots\dots + x$ $= x + x * (y-1) \}$ $\{ \text{Basis} : y = 0 \rightarrow 0$ $\text{Rekurens} : y > 0 \rightarrow x + \text{Mul}(x, \text{prec}(y)) \}$ $\text{Mul}(x,y) :$ $\text{if } y = 0 \text{ then } \{ \text{Basis } 0 \}$ $0$ $\text{else } \{ \text{Rekurens terhadap } y \}$ $x + \text{Mul}(x, \text{prec}(y))$	
<b>REALISASI VERSI-1 : REKURENS TERHADAP X</b>	
$\{ \text{Mul}(x,y) = x * y$ $= y + y + y + y \dots\dots + y$ $= y + y * (x-1) \}$ $\{ \text{Basis} : x = 0 \rightarrow 0$ $\text{Rekurens} : x > 0 \rightarrow y + \text{Mul}(\text{prec}(x), y) \}$ $\text{Mul}(x,y) :$ $\text{if } x = 0 \text{ then } \{ \text{Basis } 0 \}$ $0$ $\text{else } \{ \text{Rekurens terhadap } x \}$ $y + \text{Mul}(\text{prec}(x), y)$	

## Contoh-2 Pemangkatan bilangan bulat dengan ekspresi rekursif

### Persoalan :

Tuliskanlah definisi, spesifikasi dan realisasi sebuah fungsi yang memangkatkan sebuah integer dengan sebuah integer, dan menghasilkan sebuah integer, dengan membuat definisi rekursif dari pemangkatan.

PEMANGKATAN dua bilangan integer	Exp(x,y)
<b><u>DEFINISI DAN SPESIFIKASI</u></b>	
<b>Exp</b> : 2 <u>integer</u> > 0 → <u>integer</u> >0 { Exp (x,y) menghasilkan $x^y$ , $x \neq 0$ }	
<b><u>REALISASI</u></b>	
$\{ \text{Exp } (x,y) = x^y$ $= x * x * x * x * \dots * x$ $= x * x^{(y-1)} \}$ $\{ \text{Basis} : y = 0 \rightarrow 1$ $\text{Rekurens} : y > 0 \rightarrow x * \text{Exp}(x, \text{prec}(y)) \}$ <b>Exp</b> (x,y) : $\text{if } y = 0 \text{ then } \{ \text{Basis } 0 \}$ $1$ $\text{else } \{ \text{Rekurens terhadap } y \}$ $x * \text{Exp}(x, \text{prec}(y))$	

# LIST

## Definisi

LIST adalah **sekumpulan** elemen list yang bertype sama. Jadi list adalah koleksi objek. Elemen list mempunyai keterurutan tertentu (elemen-ke..., ada pengertian suksesor, predesesor), nilai satu elemen boleh muncul lebih dari satu kali.

## Definisi rekursif

**List** adalah sekumpulan elemen list yang bertype sama :

- Basis 0 : **list kosong** adalah sebuah **list**
- Rekurens :  
    **list** terdiri dari sebuah elemen dan sublist (yang juga merupakan **list**)

List sering disebut dengan istilah lain :

- *sequence*
- *series*

Dalam kehidupan se-hari-hari, list merepresentasi :

- teks (*list of kata*)
- kata (*list of huruf*)
- *sequential file (list of record)*
- table (*list of elemen tabel*, misalnya untuk tabel integer: *list of integer*)
- list of atom simbolik (dalam LISP)

Dalam dunia pemrograman, list merupakan suatu type data yang banyak dipakai, untuk merepresentasi objek yang diprogram misalnya :

- Antarmuka berbasis grafis (GUI): *list of Windows, list of menu item, 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*
- Dalam sistem operasi : *list of terminals, list of job*

Jika type dari elemen punya keterurutan **nilai** elemen(membesar, mengecil), dikatakan bahwa list terurut membesar atau mengecil. Bedakan keterurutan nilai ini dengan keterurutan elemen sebagai suksesor dan predesesor.

List L adalah **terurut menaik** jika dua elemen berturutan dari L yaitu e1 dan e2 (dengan e2 adalah suksesor dari e1), selalu mempunyai sifat bahwa nilai dari  $e1 \leq$  nilai dari e2.

Dari konsep type yang pernah dipelajari, elemen list dapat berupa :

- elemen yang mempunyai type sederhana (integer, karakter,...)
- elemen yang mempunyai type komposisi, mulai dari yang sederhana sampai yang kompleks
- list (rekursif !)

List kosong adalah list yang tidak mempunyai elemen, dalam notasi fungsional dituliskan sebagai [ ] .

Dua buah list disebut sama jika semua elemen list sama urutan dan nilainya.

Jika merupakan type dasar yang disediakan oleh bahasa pemrograman, maka **konstruktor** dan **selektor** list menjadi fungsi dasar yang siap dipakai seperti halnya dengan operator dasar aritmatika, relasional dan boolean.

Beberapa bahasa juga menyediakan predikat terdefinisi untuk menentukan apakah suatu list kosong atau tidak

Pada sub bab berikutnya, akan dibahas :

- List dengan elemen sederhana
- List dengan elemen karakter (teks)
- List dengan elemen bilangan integer
- List dengan elemen type bentukan (misalnya list of Point)
- List dengan elemen list (list of list)

Pembahasan hanya akan diberikan dalam bentuk definisi, kemudian relaisasi dari beberapa primitif yang penting.

## List Dengan Elemen Sederhana

**Definisi rekursif type List L dengan elemen sederhana (type dasar, type komposisi):**

- Basis 0: list kosong adalah sebuah list
- Rekurens : list dapat dibentuk dengan menambahkan sebuah elemen pada list (konstruktor), atau terdiri dari sebuah elemen dan sisanya adalah list (selektor)

Penambahan/pengambilan elemen dapat dilakukan pada “awal” list, atau pada “akhir” list. Kedua cara penambahan/pengambilan ini melahirkan konstruktor dan selektor sebagai berikut yang ditulis dalam definisi list sebagai berikut. Konsep konstruktor/selektor ini akan berlaku untuk semua list berelemen apapun.

Seperti halnya type bentukan, Konstruktor dan selektor list pada notasi fungsional hanya dibuat definisi dan spesifikasinya. Realisasinya akan diserahkan kepada bahasa pemrograman (pada bagian kedua). Bahasa fungsional biasanya menyediakan list sebagai type “primitif” dengan konstruktor/selektornya. Bahkan dalam bahasa LISP, *list* bahkan *list of list*, merupakan representasi paling mendasar dari semua representasi type lain yang mewakili koleksi.

<b>TYPE LIST</b>
<p><b><u>DEFINISI DAN SPESIFIKASI TYPE</u></b></p> <p><i>{Konstruktor menambahkan elemen di awal, notasi prefix }</i></p> <p><b>type</b> List : [ ], atau [e o List]</p> <p><i>{Konstruktor menambahkan elemen di akhir, notasi postfix }</i></p> <p><b>type</b> List : [ ] atau [List • e]</p> <p><i>{Definisi List : sesuai dengan definisi rekursif di atas }</i></p>
<p><b><u>DEFINISI DAN SPESIFIKASI KONSTRUKTOR</u></b></p> <p><b>Konso</b> : elemen, List <math>\rightarrow</math> List</p> <p><i>{Konso(e,L): menghasilkan sebuah list dari e dan L, dengan e sebagai elemen pertama e : e o L <math>\rightarrow</math> L'}</i></p> <p><b>Kons•</b> : List, elemen <math>\rightarrow</math> List</p> <p><i>{Kons(L,e): menghasilkan sebuah list dari L dan e, dengan e sebagai elemen terakhir list : L • e <math>\rightarrow</math> L'}</i></p>
<p><b><u>DEFINISI DAN SPESIFIKASI SELEKTOR O</u></b></p> <p><b>FirstElmt</b>: List tidak kosong <math>\rightarrow</math> elemen</p> <p><i>{FirstElmt(L) Menghasilkan elemen pertama list L }</i></p> <p><b>Tail</b> : List tidak kosong <math>\rightarrow</math> List</p> <p><i>{Tail(L) : Menghasilkan list tanpa elemen pertama list L, mungkin kosong }</i></p> <p><b>LastElmt</b> : List tidak kosong <math>\rightarrow</math> elemen</p> <p><i>{LastElmt(L) : Menghasilkan elemen terakhir list L }</i></p> <p><b>Head</b> : List tidak kosong <math>\rightarrow</math> List</p> <p><i>{Head(L) : Menghasilkan list tanpa elemen terakhir list L, mungkin kosong }</i></p>
<p><b><u>DEFINISI DAN SPESIFIKASI PREDIKAT DASAR</u></b></p> <p><b><u>(UNTUK BASIS ANALISA REKURENS)</u></b></p> <p><i>{Basis 0 }</i></p> <p><b>IsEmpty</b> : List <math>\rightarrow</math> <u>boolean</u></p> <p><i>{IsEmpty(L) benar jika list kosong }</i></p> <p><i>{Basis 1 }</i></p> <p><b>IsOneElmt</b>: List <math>\rightarrow</math> <u>boolean</u></p> <p><i>{IsOneElmt (X,L) adalah benar jika list L hanya mempunyai satu elemen }</i></p>

## DEFINISI DAN SPESIFIKASI PREDIKAT RELASIONAL

**IsEqual** : 2 List  $\rightarrow$  boolean

*{ IsEqual (L1,L2) benar jika semua elemen list L1 sama dengan L2: sama urutan dan sama nilainya }*

## BEBERAPA DEFINISI DAN SPESIFIKASI FUNGSI LAIN

**NbElmt** : List  $\rightarrow$  integer

*{NbElmt(L) : Menghasilkan banyaknya elemen list, nol jika kosong }*

**ElmtkeN** : integer  $\geq 0$  , List  $\rightarrow$  elemen

*{ElmtkeN (N, L) : Mengirimkan elemen list yang ke N,  $N \leq NbELmt(L)$  dan  $N \geq 0$ }*

**Copy** : List  $\rightarrow$  List

*{ Copy(L) : Menghasilkan list yang identik dengan list asal}*

**Inverse** : List  $\rightarrow$  List

*{ Inverse(L) : Menghasilkan list L yang dibalik, yaitu yang urutan elemennya adalah kebalikan dari list asal}*

**Konkat** : 2 List  $\rightarrow$  List

*{Konkat(L1,L2) : Menghasilkan konkatensi 2 buah list, dengan list L2 "sesudah" list L1}*

## BEBERAPA DEFINISI DAN SPESIFIKASI PREDIKAT

**IsMember**: elemen, List  $\rightarrow$  boolean

*{Ismember (X,L) adalah benar jika X adalah elemen list L }*

**IsFirst**: elemen, List (tidak kosong)  $\rightarrow$  boolean

*{IsFirst (X,L) adalah benar jika X adalah elemen pertama list L }*

## TYPE LIST (lanjutan)

**IsLast** : elemen, List (tidak kosong)  $\rightarrow$  boolean

*{IsLast (X,L) adalah benar jika X adalah elemen terakhir list L }*

**IsNbElmtN (N,L)** : integer  $\geq 0$  dan  $\leq NbElmt(L)$ , List  $\rightarrow$  integer

*{IsNbElmtN (N, L) true jika banyaknya elemen list sama dengan N }*

**IsInverse** : List, List  $\rightarrow$  boolean

*{IsInverse(L1,L2) true jika L2 adalah list dengan urutan elemen terbalik dibandingkan L1}*

**IsXElmtKeN** : integer  $\geq 0$  dan  $\leq N$ Elmt(L), elemen, List  $\rightarrow$  boolean

*{IsXElmtKeN (N, X,L) adalah benar jika X adalah elemen list yang ke N }*

Beberapa catatan:

- Perhatikanlah definisi beberapa fungsi yang “mirip”, dalam dua bentuk yaitu sebagai fungsi atau sebagai predikat. Misalnya :
  - LastElmt(L) dan IsLast(X,L)
  - Inverse(L) dan IsInverse(L1,L2)
  - ElmtKeN(N, L) dan IsXElmtKeN(N,X,L)

Realisasi sebagai fungsi dan sebagai predikat tersebut dalam konteks fungsional murni sangat berlebihan, namun realisasi semacam ini dibutuhkan ketika bahasa hanya mampu menangani predikat, seperti pada program logik. Bagian ini akan merupakan salah satu bagian yang akan dirujuk dalam kuliah pemrograman logik.

Berikut ini adalah realisasi dari beberapa fungsi untuk list tersebut, penulisan solusi dikelompokkan menurut klasifikasi rekurens. Untuk kemudahan pembacaan, setiap fungsi yang pernah dituliskan definisi dan realisasi nya akan diulang penulisannya, namun konstruktor, selektor dan predikat dasar untuk menentukan basis tidak dituliskan lagi.

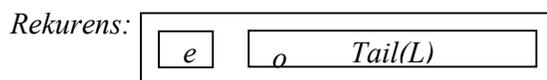
Analisa rekurens akan dituliskan melalui ilustrasi bentuk rekursif list sebagai berikut :

Basis : List kosong atau list 1 elemen

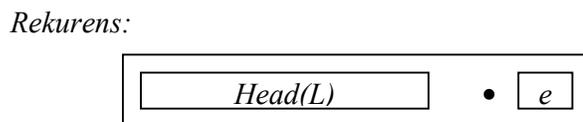
Rekurens : sebuah list terdiri dari elemen dan sisanya adalah list

Visualisasi dari list adalah sebagai “segi empat”, dan segiempat yang menggambarkan list tidak digambarkan ulang pada setiap ilustrasi

Dengan Selektor Konso, ilustrasinya adalah



Dengan Selektor Kons• ilustrasinya adalah



### Contoh 1 : Banyaknya elemen sebuah list

**Persoalan :** Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menghasilkan banyaknya elemen sebuah list. Contoh aplikasi dan hasilnya:

$NbElmt ([ ]) = 0$  ;  $NbElmt ([ a, b, c ] ) = 3$

BANYAKNYA ELEMEN	NbElmt(L)						
<b><u>DEFINISI DAN SPESIFIKASI</u></b>							
<b>NbElmt</b> : List $\rightarrow$ <u>integer</u> <i>{NbElmt(L) : Menghasilkan banyaknya elemen list, nol jika kosong }</i>							
<b><u>DEFINISI DAN SPESIFIKASI</u></b>							
<b>NElmt (L)</b> : List $\rightarrow$ <u>integer</u> <i>{NbElmt (L) menghasilkan banyaknya elemen list L }</i>							
<b><u>REALISASI: DENGAN KONSO</u></b>							
<i>{ Basis 0 : List kosong: tidak mengandung elemen, nbElmt ([ ]) = 0</i> <i>Rekurens:</i>							
<table border="0"><tr><td style="border: 1px solid black; padding: 2px;"><math>e</math></td><td style="padding: 0 10px;"><math>o</math></td><td style="border: 1px solid black; padding: 2px;"><math>Tail(L)</math></td></tr><tr><td colspan="3" style="text-align: center;"><math>1 + NbElmt ( Tail(L) )</math></td></tr></table>		$e$	$o$	$Tail(L)$	$1 + NbElmt ( Tail(L) )$		
$e$	$o$	$Tail(L)$					
$1 + NbElmt ( Tail(L) )$							
<b>NbElmt (L)</b> :							
<u>if</u> IsEmpty(L) <u>then</u> {Basis 0} 0 <u>else</u> {Rekurens} 1 + NbElmt (Tail (L))							

### Contoh 2: keanggotaan elemen

**Persoalan :** Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah predikat yang memeriksa apakah sebuah elemen x merupakan anggota dari list. Contoh aplikasi dan hasilnya:

$IsMember (x, [ ]) = \underline{false}$ ;  $IsMember (x, [a, b, c] ) = \underline{false}$

$IsMember (b, [a, b, c] ) = \underline{true}$

KEANGGOTAAN	IsMember(x,L)						
<b><u>DEFINISI DAN SPESIFIKASI</u></b>							
<b>IsMember (x,L)</b> : elemen, List $\rightarrow$ <u>boolean</u> <i>{ IsMember (x,L) true jika x adalah elemen dari list L }</i>							
<b><u>REALISASI VERSI-2 : DENGAN KONSO</u></b>							
<i>{ Basis 0 : List kosong: tidak mengandung apapun, <math>\rightarrow false</math></i> <i>Rekurens:</i>							
<table border="0"><tr><td style="border: 1px solid black; padding: 2px;"><math>e</math></td><td style="padding: 0 10px;"><math>o</math></td><td style="border: 1px solid black; padding: 2px;"><math>Tail(L)</math></td></tr><tr><td colspan="3" style="text-align: center;"><math>? x</math></td></tr></table>		$e$	$o$	$Tail(L)$	$? x$		
$e$	$o$	$Tail(L)$					
$? x$							
<i>Kasus:</i>							
$e=x \rightarrow true$ $e \neq x \rightarrow x \text{ adalah anggota } Tail(L) \}$							
<b>IsMember (x, L)</b> :							
<u>if</u> IsEmpty(x) <u>then</u> {Basis 0} <u>false</u> <u>else</u> {Rekurens : analisa kasus} <u>if</u> FirstElmt (L)=x <u>then</u> <u>true</u>							

```

else IsMember(x, Tail (L))
IsMember (x, L) :
  if IsEmpty(x) then {Basis 0}
  false
  else {Rekurens: ekspresi boolean }
  FirstElmt(L)=x or else IsMember(x, Tail (L))

```

### **REALISASI VERSI-2 : DENGAN KONS•**

{ Basis 0: list kosong tidak mengandung apapun,  $\rightarrow$  false

Rekurens:

```

  [ Head(L) ] • [ e ]
                x
                e=x  $\rightarrow$  true
                e $\neq$ x  $\rightarrow$  elemen Head(L) }

```

```

IsMember (x, L) :
  if IsEmpty(x) then {Basis 0}
  false
  else {Rekurens: analisa kasus }
  if LastElmt(L)=x then true
  else IsMember (x, Head (L))

```

### **Contoh 3 : menyalin sebuah list**

#### **Persoalan :**

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menghasilkan salinan (copy) dari sebuah list, yaitu semua elemennya sama. Contoh aplikasi dan hasilnya adalah :

Copy ([ ]) = [ ]; Copy ([a, b, c] ) = [a, b, c]

MENYALIN LIST	Copy(L)
<b>DEFINISI DAN SPESIFIKASI</b>	
Copy : List $\rightarrow$ List	
{Copy (L) menghasilkan salinan list L , artinya list lain yang identik dengan L}	
<b>REALISASI: DENGAN KONSO</b>	
{ Basis 0 : list kosong: hasilnya list kosong	
Rekurens:	
<pre>   [ e ] o [ Tail(L) ]                 e o Copy( Tail(L)) } </pre>	
Copy (L) :	
<pre>   if IsEmpty(L) then {Basis 0}   [ ]   else {Rekurens}   Konso (FirstElmt(L) , Copy(Tail (L)) </pre>	

#### Contoh 4: Membalik sebuah list

**Persoalan** : Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menerima sebuah list, dan menghasilkan list yang urutan elemennya terbalik dibandingkan urutan elemen pada list masukan. Contoh aplikasi dan hasilnya:

Inverse ([ ]) = [ ]; Inverse ([a, b, c]) = [c, b, a]

MEMBALIK LIST	Inverse(L)			
<b>DEFINISI DAN SPESIFIKASI</b>				
<b>Inverse (L)</b> : List $\rightarrow$ List <i>{Inverse (L) menghasilkan salinan list L dengan urutan elemen terbalik}</i>				
<b>REALISASI: DENGAN KONSO</b>				
<i>{ Basis 0: list kosong: hasilnya list kosong</i> <i>Rekurens:</i>				
<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td style="padding: 2px 10px;"><math>e</math></td><td style="padding: 2px 10px;"><math>o</math></td><td style="padding: 2px 10px;"><math>Tail(L)</math></td></tr></table>		$e$	$o$	$Tail(L)$
$e$	$o$	$Tail(L)$		
<i>Hasil pembalikan adalah Tail(L) • e }</i>				
<b>Inverse (L)</b> :				
if IsEmpty(L) then {Basis 0}				
[]				
else {Rekurens}				
Kons•( Inverse(Tail (L), FirstElmt(L))				

#### Contoh 5: Kesamaan dua buah list

**Persoalan** : Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah predikat yang memeriksa apakah dua buah elemen list sama. Dua buah list sama jika semua elemennya sama, dan urutan kemunculannya sama.. Contoh aplikasi dan hasilnya:

IsEqual ([], [ ]) = true ; IsEqual ([], [a ]) = false;

IsEqual([a,b,c], [a, b, c]) = true

KESAMAAN	IsEqual (L1,L2)
<b>DEFINISI DAN SPESIFIKASI</b>	
<b>IsEqual</b> : 2 List $\rightarrow$ <u>boolean</u>	
<i>{ IsEqual (L1,L2) true jika dan hanya jika kedua list sama, yaitu setiap elemen dan urutan kemunculannya sama}</i>	
<i>{ Basis 0:</i>	
IsEqual ([],[ ]) = <u>true</u>	
IsEqual ([], e o L) = <u>false</u>	
IsEqual (e o K, [ ]) <u>false</u> }	
<i>{ Rekurens : terhadap kedua list</i>	
IsEqual (e1 o K1, e2 o K2) = (e1 = e2) dan K1 = K2 }	
<b>REALISASI VERSI-1</b>	
<b>IsEqual</b> (L1,L2): {jika hanya salah satu yang kosong, maka false}	
depend on L1, L2	
IsEmpty(L1) and IsEmpty(L1) : <u>true</u> {Basis }	
IsEmpty(L1) and not IsEmpty(L1) : <u>false</u> {Basis }	
not IsEmpty(L1) and IsEmpty(L1) : <u>false</u> {Basis }	
not IsEmpty(L1) and not IsEmpty(L1) : {Rekurens }	
((FirstElmt(L1) = FirstElmt(L2)) and then	
<b>IsEqual</b> (Tail (L1), Tail (L2))	

**REALISASI VERSI-2 (CEK KEBENARANNYA!!)**

```
IsEqual (L1,L2) :
  (IsEmpty(L1) and IsEmpty(L2)) Or else
  ((FirstElmt(L1) = FirstElmt(L2)) and then
    IsEqual (Tail(L1), Tail(L2))
```

**REALISASI VERSI-3(berdasarkan banyaknya elemen)**

{Didefinisikan fungsi antara: CekEqual (L1,L2) yang akan mengecek kesamaan dua buah list yang panjangnya sama}

```
IsEqual (L1,L2) :
  if (NBElt(L1) ≠ NBElt(L2)) then
    false
  else
    CekEqual (L1,L2)
CekEqual (L1,L2) : FirstElmt(L1)= FirstElmt(L2) and then
  CekEqual (Tail(L1),Tail(L2))
```

**Contoh 6: Elemen ke N sebuah list**

**Persoalan :** Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menghasilkan elemen ke N dari sebuah list. N lebih besar dari Nol, dan list tidak boleh kosong dan  $N \leq$  banyaknya elemen list. . Contoh aplikasi dan hasilnya:

ElmtKeN (0, [] ) = tidak terdefinisi karena list kosong tidak dapat menghasilkan elemen; maka analisa rekurens harus berbasis satu dan list tidak kosong

ElmtKeN (1, [a,b,c] ) = a;

Rekurens dilakukan terhadap N, dan juga terhadap list:

<b>ELEMEN KE N</b>	<b>ElmtKeN(N,L)</b>
--------------------	---------------------

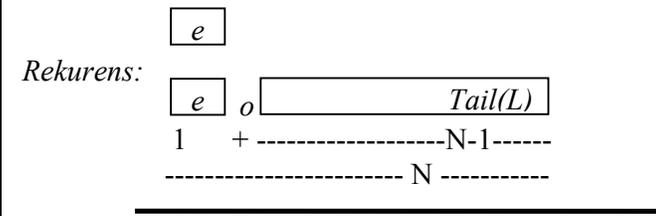
**DEFINISI DAN SPESIFIKASI**

**ElmtKeN (N,L) :** integer  $\geq$  ,List tidak kosong  $\rightarrow$  elemen

*{ElmtKeN (L) menghasilkan elemen ke-N list L,  $N \geq 0$ , dan  $N \leq$  banyaknya elemen list. }*

**REALISASI: DENGAN KONSO**

*{ Basis 1 : List dengan satu elemen , dan  $N=1$  : elemen pertama list*



*Kasus :  $N=1$  maka e*

*$N>1$  : bukan e, tetapi ElmtKeN (N-1, Tail(L))*

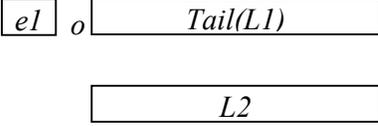
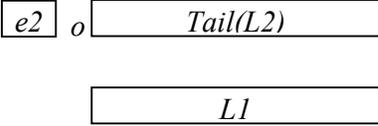
*}*

```
ElmtKeN (N,L) :
  if N=1 {Basis 1} then
    FirstElmt (L)
  else {Rekurens}
    ElmtKeN (prec (N) , Tail (L))
```

### Contoh 7: Konkatenasi dua buah list

#### Persoalan :

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menerima dua buah list, dan menghasilkan konkatenasi dari kedua list tersebut. Contoh aplikasi dan hasilnya:  
Concat ([ ], [ ]) = [ ]; Concat ([a], [b,c]) = [a,b,c]

KONKATENASI	Concat(L1,L2)
<b>DEFINISI DAN SPESIFIKASI</b>	
<b>Concat (L1,L2) : List → List</b> {Concat (L1,L2) menghasilkan konkatenasi list L1 dan L2}	
<b>REALISASI : REKURENS TERHADAP L1</b>	
{ Basis 0: L1 [] : L2 Rekurens:	
	
List Hasil: e1 o Hasil konkatenasi dari Tail(L1) dengan L2 <b>Konkat (L1, L2) :</b> if IsEmpty(L1) then {Basis 0} L2 else {Rekurens} Konso(FirstElmt(L1), Konkat(Tail(L1),L2) )	
<b>REALISASI : REKURENS TERHADAP L2</b>	
{ Basis 0: L2 [] : L1 Rekurens:	
	
List Hasil: e2 o Hasil konkatenasi dari Tail(L1) dengan L2 <b>Konkat (L1, L2) :</b> if IsEmpty(L2) then {Basis 0} L1 else {Rekurens} Konkat (Kons•(L1, FirstElmt(L2)) , Tail(L2) )	

### Contoh 8: Apakah banyaknya elemen sebuah list adalah N

**Persoalan :** Tuliskanlah definisi, spesifikasi dan relaisasi sebuah predikat yang memeriksa apakah banyaknya elemen sebuah list = N, dengan N>=0. Contoh aplikasi dan hasilnya:

IsNBElmtN (0,[ ]) = true ; IsNBElmtN (3,[a,b,c]) = true;

IsNBElmtN (0,[a]) = false

Rekurens dilakukan terhadap N.

BANYAKNYA ELEMEN	IsNbElmtN(L)
<b>DEFINISI DAN SPESIFIKASI</b>	
<b>IsNbElmtN (L,N)</b> : List, <u>integer</u> $\geq 0 \rightarrow$ <u>integer</u> <i>{IsNbElmtN (L,N) true jika banyaknya elemen list sama dengan N }</i>	
<b>REALISASI: DENGAN KONSO</b>	
<i>{ Basis 0: List kosong: false</i> <i>Rekurens:</i> <div style="display: flex; align-items: center; margin-left: 20px;"> <div style="border: 1px solid black; padding: 2px 5px; margin-right: 5px;">e</div> <span style="margin: 0 5px;">o</span> <div style="border: 1px solid black; padding: 2px 5px; margin-left: 20px;">Tail(L)</div> </div> <i># elemen: 1 (N-1) }</i>  <b>IsNbElmtN (L, N)</b> : <pre> if N=0 then {Basis 0}            if IsEmpty(L) then true else false else {Rekurens }       IsNbElmtN(Tail (L), Prec(N)) </pre>	
<b>REALISASI VERSI-2</b>	
<i>{ Realisasi ini memanfaatkan fungsi NbElmt(L) yang sudah didefinisikan }</i> <b>IsNbElmtN (L, N)</b> : NbElmt (L) =N	

**Contoh 9: Apakah X adalah Elemen ke N sebuah list**

**Persoalan :** Tuliskanlah sebuah predikat yang memeriksa apakah X adalah elemen ke N dari sebuah list. N lebih besar dari Nol, dan list tidak boleh kosong. N positif, dan bernilai 1 s/d banyaknya elemen list

APAKAH X ELEMEN KE N	IsXElmtKeN(X,N,L)
<b>DEFINISI DAN SPESIFIKASI</b>	
<b>IsXElmtKeN (N,L)</b> : <u>elemen</u> , <u>integer</u> $\geq 0$ , List (tidak kosong) $\rightarrow$ <u>boolean</u> <i>{IsXElmtKeN (L) true jika X adalah elemen ke-N list L, N <math>\geq 0</math>, dan N <math>\leq</math> banyaknya elemen list false jika tidak}</i>	
<b>REALISASI: DENGAN KONSO</b>	
<i>{ Basis 0: List dengan satu elemen , dan N=1 dan e=X : true</i>  <div style="display: flex; align-items: center; margin-left: 20px;"> <div style="border: 1px solid black; padding: 2px 5px; margin-right: 5px;">e</div> </div> <i>Rekurens:</i> <div style="display: flex; align-items: center; margin-left: 20px;"> <div style="border: 1px solid black; padding: 2px 5px; margin-right: 5px;">e</div> <span style="margin: 0 5px;">o</span> <div style="border: 1px solid black; padding: 2px 5px; margin-left: 20px;">Tail(L)</div> </div> <div style="margin-left: 20px; margin-top: 5px;"> e =X -----N-1-----  -----N -----  <hr style="width: 30%; margin-left: 0;"/> </div> <i>IsXElmtKeN (X,N-1, Tail(L))</i>	

```

IsXElmtKeN(X,N,L) :
  if IsMember (X,L) then {Analisa kasus }
    if N=1 and FirstElmt(L)=X then {Basis 0}
      true
    else {Rekurens}
      false or IsXElmtKeN(X,prec(N),Tail (L))
    else {Bukan member, pasti } false

```

### **REALISASI:**

```

{ Realisasi ini memanfaatkan fungsi ElmtKeN(L) yang sudah didefinisikan }
IsXElmtKeN(X,L,N) :
  ElmtKeN(N,L)=X

```

### **Contoh 10: Apakah inverse**

#### **Persoalan :**

Tuliskanlah definisi, spesifikasi dan realisasi dari dari sebuah predikat yang memeriksa apakah sebuah list adalah inverse dari list lain

<b>APAKAH INVERSE</b>	<b>IsInverse(L1,L2)</b>
<b>DEFINISI DAN SPESIFIKASI</b>	
<b>IsInverse (L1,L2) :</b> 2 List → <u>boolean</u> <i>{IsInverse (L1,L2) true jika L2 adalah list dengan urutan elemn terbalik dibandingkan L1, dengan perkataan lain adalah hasil inverse dari L1}</i>	
<b>REALISASI: DENGAN NAMA DAN FUNGSI ANTARA</b>	
<b>IsInverse (L) :</b> IsEqual (L3,L2)	
<b>REALISASI LAIN</b>	
<i>{ Basis 1 : list dengan satu elemen : true</i> <i>Rekurens: dua buah list sama, jika panjangnya sama dan</i>	
$L1 : \boxed{e1} \circ \boxed{\text{Tail}(L1) - x1} \bullet \boxed{x1}$	
$L2 : \boxed{e2} \circ \boxed{\text{Tail}(L2) - x2} \bullet \boxed{x2}$	
$e1=x2 \text{ dan } \text{Tail}(L1) - x1 = \text{Tail}(L2) - x2$	
<b>IsInverse (L) :</b> <u>if</u> NbElmt(L1) = NbElmt(L2) <u>then</u> {Analisa kasus } <u>if</u> IsEmpty(L1) and IsEmpty(L2) <u>then</u> {Basis 0} <u>true</u> <u>else</u> {Rekurens } ( FirstElmt(L1)=LastElmt(L2)) <u>and</u> IsInverse (Head(Tail(L1)), Head(Tail(L2))) <u>else</u> {panjang tidak sama, pasti bukan hasil inverse } <u>false</u>	

## List of Character (Teks)

Teks adalah list yang elemennya terdiri dari karakter. Karakter adalah himpunan terhingga dari 'a'..'z', 'A'..'Z', '0'..'9'

### Definisi rekursif

- Basis 0 : Teks kosong adalah teks
- Rekurens : Teks dapat dibentuk dengan menambahkan sebuah karakter pada teks

## TYPE LIST OF CHARACTER

### DEFINISI DAN SPESIFIKASI TYPE

**type** Teks : List of character

*{Definisi Teks : sesuai dengan definisi rekursif di atas }*

### DEFINISI DAN SPESIFIKASI KONSTRUKTOR

**Konso** : character, Teks → Teks

*{Konso(e,T): menghasilkan sebuah list dari e dan T, dengan e sebagai elemen pertama e :*

*e o T → T'}*

**Kons•** : Teks, character → Teks

*{Kons(T,e): menghasilkan sebuah list dari T dan e, dengan e sebagai elemen terakhir teks :*

*T • e → T'}*

### DEFINISI DAN SPESIFIKASI SELEKTOR

**FirstChar**: Teks tidak kosong → character

*{FirstElmt(L) Menghasilkan character pertama Teks T }*

**Tail** : Teks tidak kosong → Teks

*{Tail(T) : Menghasilkan list tanpa elemen pertama Teks T }*

**LastChar** : Teks tidak kosong → character

*{LastElmt(T) : Menghasilkan character terakhir Teks T }*

**Head** : Teks tidak kosong → Teks

*{Head(T) : Menghasilkan list tanpa elemen terakhir Teks T }*

### DEFINISI DAN SPESIFIKASI PREDIKAT DASAR

#### (UNTUK BASIS ANALISA REKURENS}

*{Basis 0 }*

**IsEmpty** : Teks → boolean

*{IsEmpty(T) benar jika list kosong }*

*{Basis 1 }*

**IsOneElmt**: Teks → boolean

*{IsOneElmt (X,T) adalah benar jika teks hanya mempunyai satu karakter }*

## Contoh-1 Teks : Hitung A

### Persoalan :

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menghitung kemunculan huruf 'a' pada suatu teks.

Hitung A	nba (T)
<b><u>DEFINISI DAN SPESIFIKASI</u></b>	
<b>nba</b> : Teks $\rightarrow$ <u>integer</u> $\geq 0$ { nba(T) menghasilkan banyaknya kemunculan 'a' dalam teks T }	
<b><u>REALISASI VERSI-1 : DENGAN KONS•</u></b>	
{ Basis 0: teks kosong: tidak mengandung 'a', nba ([]) = 0 Rekurens: $\boxed{\text{Head}(T)} \bullet \boxed{e}$ $a$ ekspresikan domain berdasarkan elemen lain, selain Basis nba (awt•lastchar) bisa dievaluasi kalau nba (awt) diketahui : nba(T)=nba(Head(T)) + 1 jika e adalah 'a' nba(T) =nba(Head(T)) jika e bukan 'a'} <b>nba</b> (x) : <u>if</u> IsEmpty(x) <u>then</u> {Basis 0} 0 <u>else</u> {Rekurens} nba (Head (x)) + <u>if</u> (Lastchar (x) ='a' <u>then</u> 1 <u>else</u> 0	
<b><u>REALISASI VERSI-2 : DENGAN KONSO</u></b>	
{ Basis 0 : teks kosong: tidak mengandung 'a', nba ([]) = 0 Rekurens: $\boxed{e} \circ \boxed{\text{Tail}(T)}$ $e = a$ ekspresikan domain berdasarkan elemen lain, selain Basis nba(T)= 1 + nba(Tail(T)) jika e adalah 'a' nba(T) = nba(Tail(T)) jika e bukan 'a'} <b>nba</b> (x) : <u>if</u> IsEmpty(x) <u>then</u> {Basis 0} 0 <u>else</u> {Rekurens} <u>if</u> (Firstchar (x) ='a' <u>then</u> 1 <u>else</u> 0) + nba(Tail(x))	
<b><u>REALISASI VERSI-3 : BASIS BUKAN TEKS KOSONG</u></b>	
{ Basis 1:teks dengan satu karakter : $\boxed{e}$ $a$ jika karakter adalah a, maka 1. Jika bukan 'a', maka 0	
Rekurens: $\boxed{\text{Head}(T)} \bullet \boxed{e1} \bullet \boxed{e2}$ $a$ ekspresikan domain berdasarkan elemen lain, selain Basis nba (awt . e1 . e2) =	

```

nba (awt . e1) + if (e2 = 'a')
    then 1 else 0
    selalu terhadap teks tidak kosong. }
nba (T) :
    if lastChar(T) = 'a' then 1 else
        if IsEmpty (Head(T)) then 0
        else nba (Head(T))

```

### Contoh-2 Teks : Banyaknya kemunculan suatu karakter pada teks T

#### Persoalan :

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menghitung banyaknya kemunculan sebuah karakter dalam teks.

KEMUNCULAN SUATU KARAKTER	nbx(C,T)
<b><u>DEFINISI DAN SPESIFIKASI</u></b>	
nbx : <u>character</u> , Teks → <u>integer</u> <i>{nbx(C,T) menghitung banyaknya kemunculan karakter C pada teks T}</i>	
<b><u>REALISASI</u></b>	
<i>{Basis 0}</i> (1) nbx (C, []) = 0 <i>{ teks kosong tidak mengandung karakter apapun }</i> <i>{Rekurensurence }</i> (2) nbx (C, T • e) : <i>{ jika e = C maka, 1 + kemunculan karakter pada T }</i> <i>{ jika e ≠ C, maka kemunculan karakter pada T }</i>  nbx (C,T) : <u>if</u> IsEmpty(T) <u>then</u> {Basis 0 } 0 <u>else</u> {Rekurens} <u>if</u> LastChar(T)= C <u>then</u> 1 + nbx (C, Head(T)) <u>else</u> nbx(C, Head(T))	

### Contoh-3 Teks : Banyaknya kemunculan suatu karakter pada teks T

#### Persoalan :

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah predikat yang memeriksa apakah sebuah karakter muncul dalam teks minimal N kali.

APAKAH MINIMAL ADA N KARAKTER C	Atleast(C,N,T)
<b><u>DEFINISI DAN SPESIFIKASI</u></b>	
Atleast : <u>integer</u> >0, <u>character</u> , teks → <u>boolean</u> <i>{ Atleast (N,C,T) benar, jika karakter C minimal muncul N kali pada Teks T }</i>	
<b><u>REALISASI</u></b>	
<i>{Basis 0}</i>	

(1)  $Atleast(0, C, []) = \underline{true}$   
 { karakter C pada teks kosong akan muncul 0 kali}  
 (2)  $Atleast(0, C1, C2oT) = \underline{true}$   
 { karakter apapun minimal muncul 0 kali pada teks selalu benar }  
 (3)  $Atleast(1+N, C1, []) = \underline{false}$ , N bil. natural.  
 {Rekurens}

(4)  $Atleast(1+n, C1, C2oT) = \text{if } (C1=C2 \text{ then } Atleast(n, C1, T) \text{ else } Atleast(1+n, C1, T)$

```

Atleast (n, C, T):
  if IsEmpty(T) then {Basis 0}
    0
  else {Rekurens :analisa kasus}
    depend on n
      n = 0 : true
      n > 0 : if (C = FirstChar(T))
        then Atleast (n, C, Tail(T))
        else Atleast (1+n, C, Tail(T))
  
```

#### Contoh-4 Teks : Palindrom

##### Persoalan :

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah predikat yang memeriksa apakah sebuah teks palindrome. Sebuah teks disebut sebagai palindrom jika dibaca dari awal sampai dengan terakhir identik dengan dari karakter terakhir sampai ke awal. Contoh teks palindrom : “NABABAN”, “KASUR RUSAK”, “KASUR NABABAN RUSAK”

Memeriksa palindrome	IsPalindrome (T)
<b><u>DEFINISI DAN SPESIFIKASI</u></b>	
<b>IsPalindrome</b> : Text → <u>boolean</u>	
<i>{IsPalindrome(T) benar jika teks T adalah palindrome : jika dibaca dari kiri ke kanan, hasilnya sama dengan jika dibaca dari kanan ke kiri }</i>	
<b><u>REALISASI DENGAN O</u></b>	
<i>{ Basis 0 : teks kosong adalah palindrome teks dengan satu elemen adalah palindrom</i>	
<i>Rekurens :</i>	
<p style="text-align: center;">           ----- T -----  <div style="display: flex; align-items: center; justify-content: center;"> <div style="border: 1px solid black; padding: 2px 5px; margin-right: 10px;">e</div> <div style="border: 1px solid black; width: 100px; height: 15px; margin-right: 10px;"></div> <div style="margin-right: 10px;">•</div> <div style="border: 1px solid black; padding: 2px 5px; margin-left: 10px;">e'</div> </div>             ----- Tail(T) -----         </p>	
}	
<b>IsPalindrome</b> (T) :	
<u>depend on</u> (T)	
{Kasus khusus } T = [] : <u>true</u>	
{Basis-1} T = [e] : <u>true</u>	
{Rekurens} <u>else</u> : (FirstChar(T) = LastChar(T)) <u>and then</u> IsPalindrome (Head(Tail(T)))	

## List of Integer

### Definisi rekursif list integer

- Basis: List kosong adalah list bilangan integer
- Rekurens : list bilangan integer dibuat dengan cara menambahkan sebuah integer pada list bilangan integer.

### TYPE LIST INTEGER

#### DEFINISI DAN SPESIFIKASI TYPE

**type** List of integer

*{Definisi: list yang elemennya integer, sesuai dengan definisi rekursif di atas }*

#### DEFINISI DAN SPESIFIKASI KONSTRUKTOR

**Konso** : integer, List of integer tidak kosong  $\rightarrow$  List of integer

*{Konso(e,L): menghasilkan sebuah list dari e dan L, dengan e sebagai elemen pertama e :*

*e o L  $\rightarrow$  L'}*

**Kons•** : List of integer tidak kosong, integer  $\rightarrow$  List of integer

*{Kons(L,e): menghasilkan sebuah list dari L dan e, dengan e sebagai elemen terakhir list :*

*L • e  $\rightarrow$  L'}*

#### DEFINISI DAN SPESIFIKASI SELEKTOR

**FirstElmt**: List of integer tidak kosong  $\rightarrow$  integer

*{FirstElmt(L) Menghasilkan elemen pertama list L }*

**Tail** : List of integer tidak kosong  $\rightarrow$  List of integer

*{Tail(L) : Menghasilkan list tanpa elemen pertama list L }*

**LastElmt** : List of integer tidak kosong  $\rightarrow$  integer

*{LastElmt(L) : Menghasilkan elemen terakhir list L }*

**Head** : List of integer tidak kosong  $\rightarrow$  List of integer

*{Head(L) : Menghasilkan list tanpa elemen terakhir list L }*

#### DEFINISI DAN SPESIFIKASI PREDIKAT DASAR

##### (UNTUK BASIS ANALISA REKURENS}

*{Basis 0 }*

**IsEmpty** : List of integer  $\rightarrow$  boolean

*{IsEmpty(L) benar jika list kosong }*

*{Basis 1 }*

**IsOneElmt**: List of integer  $\rightarrow$  boolean

*{IsOneElmt (X,L) adalah benar jika teks hanya mempunyai satu elemen }*

### **DEFINISI DAN SPESIFIKASI PREDIKAT KEABSAHAN**

**IsListInt**: List  $\rightarrow$  boolean

*{IsListInt(Lmenghasilkan true jika L adalah list dengan elemen integer }*

**Berikut ini diberikan contoh persoalan yang spesifik terhadap pemrosesan elemen list yang bertype bilangan integer**

#### **Contoh-1 List integer : Maksimum**

##### **Persoalan :**

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menghasilkan elemen bernilai maksimum dari sebuah list bilangan integer. Perhatikanlah bahwa fungsi rekursif untuk maksimum didasari atas basis 1, sebab jika list kosong, maka nilai maksimum tidak terdefinisi. Untuk ini predikat dasar untuk mentest adalah predikat IsOneElmt , basis 1

### **NILAI MAKSIMUM LIST INTEGER**

**maxlist(Li)**

#### **DEFINISI DAN SPESIFIKASI**

**maxlist** : list of integer tidak kosong  $\rightarrow$  integer

*{ maxlist(Li) : menghasilkan elemen Li yang bernilai maksimm }*

**IsOneElmt** : list of integer tidak kosong  $\rightarrow$  boolean

*{OneElmt(L) benar, jika list L hanya mempunyai satu elemen }*

#### **REALISASI**

```
maxlist(Li) :  
    if IsOneElmt( Li) then {Basis 1}  
        LastElmt (Li)  
    else {Rekurens}  
        max2 (LastElmt (Li), maxList (Head (Li))
```

*{dengan max2(a,b) adalah fungsi yang mengirimkan nilai maksimum dari dua buah integer a dan b yang pernah dibahas pada bagian I. }*

## Contoh-2 List integer : Ukuran (Dimensi) List

### Persoalan :

Tuliskanlah definisi, spesifikasi dan realisasi dari dari sebuah fungsi yang menghasilkan banyaknya elemen (dimensi) sebuah list integer

UKURAN LIST	Dimensi(Li)
<b><u>DEFINISI</u></b> Dimensi : <u>list of integer</u> $\rightarrow$ <u>integer</u> $> 0$ { DIM(Li) menghasilkan banyaknya elemen list integer } { Basis 0 : dimensi list kosong = 0 Rekurens : dimensi (Li) $\boxed{e} \quad 0 \quad \boxed{\text{Tail}(Li)}$ $1 \quad + \quad \text{Dimensi}(\text{Tail}(Li))$	
<b><u>REALISASI</u></b> $\text{Dimensi} (Li) :$ $\text{if } \text{IsEmpty}( Li) \text{ then } 0 \text{ else } 1 + \text{Dimensi}(\text{Tail}(Li))$	

Catatan : ukuran list ini juga berlaku untuk list dengan elemen apapun. Namun karena akan dipakai untuk operasi lainnya, maka direalisasi.

### Contoh-3 List integer : Penjumlahan dua buah list integer:

#### Persoalan :

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menjumlahkan dua buah list integer dan menghasilkan sebuah list integer.

Perhatikanlah bahwa contoh ini adalah contoh rekurens terhadap kedua list.

PENJUMLAHAN DUA LIST INTEGER	List+(L1,L2)
<b><u>DEFINISI</u></b>	
<b>List+</b> : 2 <u>list of integer</u> $\geq 0 \rightarrow$ <u>list of integer</u> $\geq 0$	
<i>{ List+ (Li1,Li2): menjumlahkan setiap elemen list integer yang berdimensi sama, hasilnya berupa list integer berdimensi tsb. }</i>	
<i>{ Basis 0 : Dimensi(Li1)=0 and Dimensi(Li2)= 0 : []</i>	
<i>Rekurens : Dimensi(Li1) = 0 and Dimensi(Li2) = 0 :</i>	
$e1$ o <span style="border: 1px solid black; padding: 2px;">Tail(Li1)</span>	
$e2$ o <span style="border: 1px solid black; padding: 2px;">Tail(Li2)</span>	+
<hr/>	
$e1+e2$ o <i>List+(Tail(Li1), Tail(Li2))</i>	
}	
<b><u>REALISASI</u></b>	
<b>List+</b> (Li1, Li2) :	
<u>if</u> Dimensi(Li1) = 0 <u>then</u> {Basis 0}	
[]	
<u>else</u> {Rekurens}	
Konso(FirstElmt (Li1)+FirstElmt (Li2),	
List+(Tail (Li1),Tail (Li2) )	

#### Contoh-4 List of integer : INSERTION SORT

Persoalan :

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menghasilkan list terurut **membesar** dari sebuah list bilangan integer sembarang dengan metoda “insertion sort”

INSERTION SORT	Insert(Li)
<p><b>DEFINISI</b></p> <p><b>Insert</b> : <u>list of integer</u> → <u>list of integer</u> { <i>Insert(Li)</i> menghasilkan list integer yang terurut dengan metoda insertion sort } { <i>Basis 0</i> : list kosong sudah terurut, hasilnya list kosong <i>Rekurens</i> : <i>Insert (Li)</i></p> <p style="text-align: center;"><span style="border: 1px solid black; padding: 2px;">e</span> o <span style="border: 1px solid black; padding: 2px;">Tail(Li)</span></p> <p><i>Hasil: Insert e ke Tail(Li) yang sudah terurut dg Insertion sort }</i></p> <p><b>Insert</b> : <u>integer</u>, <u>list of integer tidak kosong</u>, terurut membesar → <u>list of integer</u> terurut membesar { <i>Insert(x,Li)</i> melakukan insert x ke Li menghasilkan list terurut membesar { <i>Basis 0</i> : list kosong, insert sebuah elemen x ke list kosong : [x] <i>Rekurens</i> :</p> <p style="text-align: center;"><span style="border: 1px solid black; padding: 2px;">e</span> o <span style="border: 1px solid black; padding: 2px;">Tail(Li)</span></p> <p style="text-align: center;"><math>x \leq e : x \text{ o } Li</math> <math>x &gt; e : e \text{ o } \text{Insert}(x, \text{Tail}(Li))</math></p> <p><i>Catatan : insert tidak pernah “merusak” keterurutan elemen list!}</i></p>	
<p><b>REALISASI</b></p> <pre>Insert (x,Li) :   if IsEmpty(Li) then {Basis 0}     [x]   else {Rekurens}     if (x ≤ FirstElmt(Li) )       then Konso(x,Li)     else Konso(FirstElmt(Li),Insert (x,Tail(Li)))  Insert (Li) :   if IsEmpty( Li) then {Basis 0}     [ ]   else {Rekurens}     Insert(First(Li),Insert(Tail(Li)))</pre>	

### Contoh-5 List integer : Banyaknya kemunculan nilai maksimum

Fungsi dengan *range* type bentukan tanpa nama

#### Persoalan :

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menghasilkan nilai maksimum **dan** banyaknya kemunculan dari nilai maksimum dari sebuah list bilangan integer positif. Nilai maksimum hanya terdefinisi jika list tidak kosong. Contoh ini adalah contoh fungsi yang mengembalikan type komposisi tanpa nama dengan parameter masukan sebuah list. Perhatikanlah realisasi dalam dua versi sebagai berikut dan pelajarilah implementasinya dengan bahasa pemrograman fungsional yang dipakai. Contoh : List [11 3 4 5 11 6 8 11] menghasilkan <11,3>

Solusi versi-1 : dengan analisa rekurens berdasarkan definisi

Analisis rekurens :

Basis 1: List dengan satu elemen e: menghasilkan <e,1>

Rekurens : List dengan struktur e oTail(Li) harus dianalisis sebagai berikut :

$$\boxed{e} \circ \boxed{\text{Tail}(Li)}$$

Nilai maks= m, #kemunculam m=n

Jika m adalah nilai maksimum Tail(Li) dan n adalah banyaknya kemunculan m pada Tail(Li), maka ada tiga kemungkinan :

m < e : e adalah maksimum “baru”, maka hasilnya <e,1>

m = e : terjadi kemunculan m, maka hasilnya <m,n+1>

m > e : e dapat diabaikan, maka hasilnya <m,n>

### KEMUNCULAN MAKSIMUM (versi-1)

#### DEFINISI DAN SPESIFIKASI

**maxNb** : list of integer → <integer, integer>

{maxNb(Li) menghasilkan <nilai maksimum, kemunculan nilai maksimum> dari suatu list integer Li ; <m,n> =m adalah maks x dari n # occurence m dalam list}

#### REALISASI VERSI-1

```
MaxNb(Li) : {menghasilkan nilai maksimum dan kemunculannya }
  if OneElmt(Li) then {Basis 1}
    <FirstElmt(Li),1>
  else
    let <m,n> = MaxNb (Tail(Li))
    in   depend on m, FirstElmt(Li)
         m < FirstElmt(Li) : < FirstElmt(Li), 1>
         m = FirstElmt(Li) : < m, 1+n>
         m > FirstElmt(Li) : <m,n>
```

Versi kedua : dengan realisasi fungsi antara yang menghasilkan:

- Nilai maksimum
- Banyaknya kemunculan nilai maksimum

## **KEMUNCULAN MAKSIMUM (versi-2)**

### **DEFINISI DAN SPESIFIKASI**

**maxNb** : list of integer → <integer, integer>

{*maxNb(Li)* menghasilkan <nilai maksimum, kemunculan nilai maksimum> dari suatu list integer *Li* ; <*m,n*> =*m* adalah maks *x* dari *n* # occurrence *m* dalam list}

**max** : list of integer → integer

{*max(Li)* menghasilkan nilai maksimum dari elemen suatu list integer *Li* }

**Vmax** : list of integer → integer

{*VMax(Li)* adalah *NbOcc(max(Li),Li)* yaitu banyaknya kemunculan nilai maksimum dari *Li* , dengan aplikasi terhadap *NbOcc(max(Li),Li)*}

**NbOcc** : integer, list of integer → integer > 0

{*NbOcc(X,Li)* yaitu banyaknya kemunculan nilai *X* pada *Li* }

### **REALISASI VERSI-2**

```
max (Li) : { nilai maksimum list }
  if IsOneELmt(Li) then {Basis 1}
    [x]
  else {Rekurens : analisa kasus }
    if (FirstElmt(Li) > max(Tail(Li)))
      then FirstElmt(Li)
    else max(Tail(Li))
Vmax (Li) : { Banyaknya kemunculan nilai maks }
  NBOcc(max(Li), Li)
NbOcc(X, Li) : { banyaknya kemunculan nilai
  if IsOneELmt(Li) then {Basis 1, analisa kasus}
    if X=FirstElmt(Li) then
      1
    else
      0
  else {Rekurens : analisa kasus }
    if X=FirstElmt(Li) then
      1 + NbOcc(Tail(Li))
    else
      NbOcc(Tail(Li))
MaxNb(Li) : <max(Li), NbOcc(max(Li), Li)>
```

## Himpunan (Set)

### Definisi :

Himpunan (*set*) adalah sebuah list yang setiap elemennya hanya muncul sekali (unik). List "kosong" adalah himpunan kosong.

Contoh :

[apel, jeruk, pisang] adalah himpunan

[apel, jeruk, mangga, jeruk] bukan himpunan

<b>TYPE SET (HIMPUNAN)</b>
<b><u>DEFINISI DAN SPESIFIKASI TYPE</u></b> <i>{Set adalah List dengan tambahan syarat bahwa tidak ada elemen yang sama }</i> <i>{ Semua konstruktor, selektor dan fungsi pada List berlaku untuk Himpunan }</i>
<b><u>DEFINISI DAN SPESIFIKASI KONSTRUKTOR HIMPUNAN DARI LIST</u></b> <i>{ Himpunan dibentuk dari list }</i> <b>MakeSet</b> (L) : list → set <i>{ membuat sebuah set dari sebuah list }</i> <i>{ yaitu membuang semua kemunculan yang lebih dari satu kali }</i> <i>{ List kosong tetap menjadi list kosong }</i>
<b><u>DEFINISI DAN SPESIFIKASI PREDIKAT</u></b> <b>IsSet</b> : list → boolean <i>{ IsSet(L) true jika L adalah set }</i>  <b>IsSubSet</b> : 2 set → boolean <i>{ IsSubSet (H1,H2) true jika H1 adalah subset dari H2: semua elemen H1 adalah juga merupakan elemen H2 }</i>
<b><u>DEFINISI DAN SPESIFIKASI OPERASI TERHADAP HIMPUNAN</u></b> <b>MakeIntersect</b> : 2 set → set <i>{ Intersect (H1,H2) membuat interseksi H1 dengan H2 : yaitu set baru dengan anggota elemen yang merupakan anggota H1 dan juga anggota H2 }</i> <b>MakeUnion</b> : 2 set → set <i>{ Union (H1,H2) membuat union H1 dengan H2 : yaitu set baru dengan semua anggota elemen H1 dan anggota H2 }</i>

Untuk kasus himpunan berikut, dibutuhkan beberapa fungsi terhadap list sebagai berikut :

**IsMember** : elemen, list  $\rightarrow$  boolean

*{ IsMember(e,L) true jika e adalah elemen list L }*

**Rember** : elemen, list  $\rightarrow$  list

*{ Rember (x,L) menghapus sebuah elemen bernilai x dari list }*

*{ list yang baru berkurang SATU elemennya yaitu yang bernilai e }*

*{ List kosong tetap menjadi list kosong }*

**MultiRember** : elemen, list  $\rightarrow$  list

*{ MultiRember (x,L) menghapus semua elemen bernilai x dari list }*

*{ list yang baru tidak lagi mempunyai elemen yang bernilai x }*

*{ List kosong tetap menjadi list kosong }*

### Contoh 1: Menghapus SEBUAH elemen sebagai anggota list

Predikat ini dibutuhkan untuk membentuk himpunan

HAPUS1ELEMEN	Rember(e.L)
<b><u>DEFINISI</u></b>	
<p><b>Rember</b> : elemen, list <math>\rightarrow</math> list</p> <p><i>{ Rember (x,L) menghapus sebuah elemen bernilai x dari list }</i></p> <p><i>{ list yang baru berkurang SATU elemennya yaitu yang bernilai e }</i></p> <p><i>{ List kosong tetap menjadi list kosong }</i></p> <p><i>{ Base : list kosong : <math>\rightarrow</math> list kosong }</i></p> <p>Rekurens :</p> $\boxed{e }^x \circ \boxed{\text{Tail}(L)}$ <p><i>e = x : hasil adalah Tail(L) ,</i></p> <p><i>e <math>\neq</math> x : e  o Hasil rember(e,Tail(L)) }</i></p>	
<b><u>REALISASI</u></b>	
<pre> Rember (x, L) :     <u>if</u> IsEmpty(L) <u>then</u> {Basis }         L     <u>Else</u> {Rekurens : analisa kasus }         <u>if</u> FirstElmt(L)=x <u>then</u> Tail(L)         <u>else</u> Konso (FirstElmt(L),Rember(x,Tail(L)))                     </pre>	

**Contoh 2: Menghapus SEMUA elemen e sebagai anggota list :**  
**Predikat ini dibutuhkan untuk membentuk himpunan**

HAPUS SEMUA ELEMEN	MultiRember(e,L)
<p><b><u>DEFINISI</u></b></p> <p><b>MultiRember</b> : elemen, list → list</p> <p><i>{ MultiRember (x,L) menghapus semua elemen bernilai x dari list }</i>  <i>{ list yang baru tidak lagi mempunyai elemen yang bernilai x }</i>  <i>{ List kosong tetap menjadi list kosong }</i>  <i>{ Base : list kosong : → List kosong }</i>            Rekurens :</p> $e \quad o \quad \boxed{\text{Tail}(L)}$ <p><i>e = x : hapus semua x dari Tail(L) ,</i>  <i>e ≠ x : e l o hasil penghapusan semua x dari Tail(L)}</i></p>	
<p><b><u>REALISASI</u></b></p> <p><b>MultiRember</b> (x, L) :</p> <pre> if IsEmpty(L) then {Basis}   L else {Rekurens : analisa kasus }   if FirstElmt(L)=x   then MultiRember(x,Tail(L))   else Konso (FirstElmt(L),MultiRember(x,Tail(L)))           </pre>	

**Contoh 3: Mentest apakah sebuah list adalah himpunan**

**Persoalan:**

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah predikat yang akan mentest apakah sebuah list adalah Himpunan

APAKAH SET	IsSet(L)
<p><b><u>DEFINISI PREDIKAT</u></b></p> <p><b>IsSet</b> : list → boolean</p> <p><i>{ Set(L) true jika L adalah set }</i>  <i>{ Base : list kosong adalah set }</i>            Rekurens :</p> $e \quad o \quad \boxed{\text{Tail}(L)}$ <p><i>merupakan set jika Tail(L) tidak mengandung e}</i></p>	
<p><b><u>REALISASI VERSI-1</u></b></p> <p><b>IsSet</b> (L) :</p> <pre> if IsEmpty(L) then {Basis: list kosong adalah himpunan kosong }           </pre>	

```

    true
else {Rekurens : analisa kasus }
    if IsMember(FirstElmt(L),Tail(L)) then false
    else IsSet(Tail(L))

```

### **REALISASI VERSI-2**

```

IsSet(L) :
    if IsEmpty(S) then {Basis }
        true
    else {Rekurens:}
        not IsMember(FirstElmt(L),Tail(L)) or then IsSet(Tail(L))

```

### **REALISASI**

```

IsSet(L) :
    Isempy(S) or then not IsMember(FirstElmt(L),Tail(L))
    or then IsSet(Tail(L))

```

#### **Contoh 4: Membuat sebuah set dari sebuah list**

##### **Persoalan :**

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang akan membentuk sebuah Himpunan dari elemen-elemennya, yaitu dengan meniadakan duplikasi elemen

<b>MEMBENTUK SET (versi-1)</b>	<b>MakeSet (L)</b>
<b><u>DEFINISI</u></b>	
<p>MakeSet1 (L) : list → set</p> <p><i>{ membuat sebuah set dari sebuah list }</i>  <i>{ yaitu membuang semua kemunculan yang lebih dari satu kali}</i>  <i>{ List kosong tetap menjadi list kosong }</i>  <i>{ Base : list kosong : → List kosong }</i>                      Rekurens :</p> <div style="text-align: center;"> <math display="block">e \circ \boxed{\text{Tail}(L)}</math> </div> <p>Untuk setiap e :</p> <p style="text-align: center;"><i>e adalah Member dari Tail(L) : MakeSet(Tail(L))</i>  <i>e bukan Member dari Tail(L) : e o MakeSet(Tail(L)) }</i></p>	
<b><u>REALISASI</u></b>	
<pre> MakeSet1(L) :     if IsEmpty(L) then {Basis}         L     else {Rekurens}         if IsMember(FirstElmt(L),Tail(L)) then             MakeSet(TAIL(L))         else Konso (FirstElmt(L),MakeSet(Tail(L)) </pre>	
<b><u>APLIKASI</u></b>	
<p>⇒ MakeSet( [apel, sirsak, per, mangga, apel, jeruk, sirsak])</p> <p>{Himpunan hasil adalah : [per, mangga, apel, jeruk, sirsak] }</p>	

<b>MEMBENTUK SET (versi-2)</b>	<b>MakeSet (L)</b>
--------------------------------	--------------------

**DEFINISI**

MakeSet2 : list → set

*{ MakeSet2 (S) membuat sebuah set dari sebuah list }  
 { yaitu mempertahankan elemen pertama yang muncul, dan membuang kemunculan elemen tersebut pada sisa jika muncul lebih dari satu kali}*

*{ List kosong tetap menjadi list kosong }*

*{ Base : list kosong : → ()*

*Rekurens :*



*e o MakeSet(Tail(L)) dengan Tail(L) yg tidak lagi mengandung e}*

**REALISASI**

MakeSet2 (L) :

if IsEmpty(L) then {Basis }

L

Else {Rekurens }

Konso (FirstElmt (L) , MakeSet (MULTIReMber (FirstElmt (L) , Tail (L) )

**APLIKASI**

⇒ MakeSet( [apel, sirsak, per, mangga, apel, jeruk, sirsak])

{Himpunan hasil : [apel, sirsak, per, mangga, jeruk] }

**Contoh 5: Mentest apakah sebuah set merupakan subset dari set yang lain**

**Persoalan :**

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah predikat yang akan mentest apakah sebuah himpunan merupakan himpunan bagian dari himpunan lain yang diberikan

<b>APAKAH SUBSET</b>	<b>IsSubSet(H1,H2)</b>
----------------------	------------------------

**DEFINISI PREDIKAT**

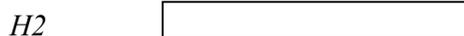
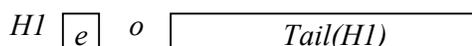
IsSubSet : 2 set → boolean

*IsSubSet (H1,H2) true jika H1 adalah subset dari H2: semua elemen H1 adalah juga merupakan elemen H2 }*

*{ List kosong adalah subset dari set apapun}*

*{ Base : list kosong : → true*

*Rekurens :*



Setiap karakter H1 harus dicek thd H2 :  
 e anggota dari H2 : adalah subset jika Tail(H1) adalah subset H2  
 e bukan anggota H2: H1 pasti bukan subset H2 }

**REALISASI**

```
IsSUBSet (H1, H2) :
{Basis} if Isemtyp(H1) then true
{Rekurens} else {analisa kasus }
           if not IsMember (FirstElmt (H1), H2) then false
           else { e anggota H2 }
           IsSubSet (Tail (H1), H2)
```

Sebagai latihan, buatlah realisasi yang hasilnya identik, namun dengan memanfaatkan ekspresi boolean dengan operator **and then** dan **or else**

**Contoh 6 :Mentest kesamaan dua buah set**

**Persoalan :**

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah predikat yang akan mentest apakah dua buah himpunan identik.

KESAMAAN DUA SET	IsEQSet (H1,H2)
<b>DEFINISI PREDIKAT</b>	
<p><b>IsEQSet</b> : 2 set → <u>boolean</u>            { IsEQSet (H1,H2) true jika H1 "sama dengan" H2, yaitu jika semua elemen H1 juga merupakan elemen H2, tanpa peduli urutannya }            { H1==H2 jika dan hanya jika H1 adalah subset H2 dan H2 adalah subset H1 }</p>	
<b>REALISASI</b>	
<p><b>IsEQSet</b> (H1, H2) : IsSUBSet (H1, H2) <u>and then</u> IsSUBSet (H2, H1)</p>	

**Contoh 7: Mentest apakah dua buah set berinterseksi**

**Persoalan :**

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah predikat yang akan mentest apakah dua buah himpunan saling beririsan

APAKAH INTERSEKSI	Intersect (H1,H2)
<b>DEFINISI PREDIKAT</b>	
<p>IsIntersect : 2 set → <u>boolean</u>            { IsIntersect (H1,H2) true jika H1 berinterseksi dengan H2 : minimal ada satu anggota yang sama. Himpunan kosong bukan merupakan himpunan yang berinterseksi dengan himpunan apapun }            { Base : Salah satu kosong : → <u>false</u>            Rekurens :</p>	

$H1 \boxed{e1} \circ \boxed{\text{Tail}(H1)}$

$H2 \boxed{\phantom{e1}} \circ \boxed{\text{Tail}(H2)}$

$e1$  adalah Member dari  $H2$  : true  
 $e1$  bukan Member dari  $H2$  :  $\text{Intersect}(\text{Tail}(H1), H2)$  }

### REALISASI

```
IsIntersect (L) :  
  depend on H1, H2  
  {Basis : }  
  Isempty(H1) and Isempty(H2)      : false  
  not Isempty(H1) and Isempty(H2)  : false  
  Isempty(H1) and not Isempty(H2)  : false  
  not Isempty(H1) and not Isempty(H2) : { Rekurens}  
  IsMember(FirstElmt(H1), H2) or then IsIntersect(Tail(H1), H2)
```

### **Contoh 8: Membuat interseksi dari dua buah set :**

#### **Persoalan :**

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menghasilkan sebuah himpunan yang elemennya adalah hasil irisan dari dua buah himpunan

<b>BUAT INTERSEKSI</b>	<b>Intersect(H1,H2)</b>
------------------------	-------------------------

### DEFINISI

MakeIntersect : 2 set  $\rightarrow$  set

{ Intersect (H1,H2) membuat interseksi H1 dengan H2 : yaitu set baru dengan anggota elemen yang merupakan anggota H1 dan juga anggota H2 }

{ Base : Jika salah satu kosong , hasilnya set kosong

Rekurens :

$H1 \boxed{e1} \circ \boxed{\text{Tail}(H1)}$

$H2 \boxed{\phantom{e1}} \circ \boxed{H2}$

$e1$  adalah Member dari  $H2$  :  $e1 \circ \text{MakeIntersect}(\text{Tail}(H1), H2)$   
 $e1$  bukan Member dari  $H2$  :  $\text{MakeIntersect}(\text{Tail}(H1), H2)$  }

### REALISASI

```
MakeIntersect (H1, H2) :  
  depend on H1, H2  
  {Basis }  
  Isempty(H1) and Isempty(H2)      : []  
  not Isempty(H1) and Isempty(H2)  : []  
  Isempty(H1) and not Isempty(H2)  : []  
  not Isempty(H1) and not Isempty(H2) : {Rekurens}  
  if IsMember(FirstElmt(H1), H2)  
  then Konso (FirstElmt(H1), MakeIntersect(Tail(H1), H2))  
  else MakeIntersect(Tail(H1), H2)
```

### Contoh 9: Membuat Union dari dua buah set

#### Persoalan :

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menghasilkan himpunan yang elemennya merupakan hasil union (gabungan) dari dua buah himpunan

SET UNION	MakeUnion (H1,H2)
<b>DEFINISI</b>	
<b>MakeUnion</b> : 2 set $\rightarrow$ set	
<i>{ MakeUnion (H1,H2) membuat union H1 dengan H2 : yaitu set baru dengan semua anggota elemen H1 dan anggota H2 }</i>	
<i>{ Base : Jika salah satu kosong, hasilnya adalah himpunan yang tidak kosong Kedua set kosong , hasilnya sebuah set yang kosong</i>	
Rekurens :	
$H1$	$e1$ o $Tail(H1)$
$H2$	$H2$
<i>e1 adalah Member dari H2 : buang e1, MakeUnion(Tail(H1),H2)</i>	
<i>e1 bukan Member dari H2 : e1 o MakeUnion(Tail(H1),H2)}</i>	
<b>REALISASI</b>	
<b>MakeUnion</b> (H1, H2) :	
<u>depend on</u> H1, H2	
{Basis }	
IsEmpty(H1) <u>and</u> IsEmpty(H2) : []	
<u>not</u> IsEmpty(H1) <u>and</u> IsEmpty(H2) : H1	
IsEmpty(H1) <u>and</u> <u>not</u> IsEmpty(H2) : H2	
<u>not</u> IsEmpty(H1) <u>and</u> <u>not</u> IsEmpty(H2) : {Rekurens}	
<u>if</u> IsMember(FirstElmt(H1),H2)	
<u>then</u> MakeUnion(Tail(H1),H2))	
<u>else</u> Konso(FirstElmt(H1),Union(Tail(H1),Tail(H2)))	

## List of List

### Definisi rekursif

List of list adalah list yang :

- mungkin kosong,
- mungkin terdiri dari sebuah elemen yang disebut **atom dan sisanya adalah list of list**,
- mungkin terdiri dari sebuah elemen berupa **list dan sisanya adalah list of list**

Jadi List of List adalah list S yang elemennya adalah list. Dalam bahasa LISP, type ini disebut sebagai S-expression

Untuk membedakan antara list dengan atom: List dituliskan di antara tanda kurung (), sedangkan Atom dituliskan tanpa tanda kurung

Contoh List :

[ ] adalah list kosong

[a , b , c ] adalah list dengan elemen berupa 3 atom

[ [ ], [a ,b ,c] , [ d ,e] ,f ] adalah :

list dengan elemen list kosong, L1, L2 dan sebuah atom

L1 adalah list dengan elemen 3 buah atom a, b, c

L2 adalah list dengan elemen 2 buah atom d,e

Untuk list khusus ini, nama Konstruktor dan Selektor tidak dibedakan dengan list yang hanya mengandung elemen dasar, namun diperlukan predikat tambahan sebagai berikut: yaitu untuk mengetahui apakah sebuah ekspresi adalah Atom atau List.

Atom dapat berupa:

- atom numerik (yang dapat dipakai sebagai operan dalam ekspresi aritmatik)
- atom simbolik

## **TYPE LIST-OF-LIST**

### **DEFINISI DAN SPESIFIKASI PREDIKAT KHUSUS UNTUK LIST OF LIST**

**IsEmpty** : list of list → boolean

*{IsEmpty(S) benar jika S adalah list of list kosong}*

**IsAtom** : list of list → boolean

*{IsAtom(S) menghasilkan true jika list adalah atom, yaitu terdiri dari sebuah atom }*

**IsList** : list of list → boolean

*{ IsList(S) menghasilkan true jika S adalah sebuah list (bukan atom)}*

### **DEFINISI DAN SPESIFIKASI KONSTRUKTOR**

**KonsLo** : List, List of list → List of list

*{ KonsLo(L,S) diberikan sebuah List L dan sebuah List of List S, membentuk list baru dengan List yang diberikan sebagai elemen pertama List of list:  $L o S \rightarrow S'$ }*

**KonsL•** : List of list , List → List of list

*{KonsL•(S,L) diberikan sebuah List of list S dan sebuah list L, membentuk list baru dengan List yang diberikan sebagai elemen terakhir list of List:  $S \bullet L \rightarrow S'$ }*

### **DEFINISI DAN SPESIFIKASI SELEKTOR**

**FirstList**: List of list tidak kosong → List

*{FirstList(S) Menghasilkan elemen pertama list, mungkin sebuah list atau atom }*

**TailList** : List of list tidak kosong → List of list

*{TailList(S) Menghasilkan "sis" list of list S tanpa elemen pertama list S }*

**LastList** : List of list tidak kosong → List of list

*{LastList(S) : Menghasilkan elemen terakhir list of list S, mungkin list atau atom }*

**HeadList** : List of list tidak kosong → List of list

*{HeadList(S) Menghasilkan "sis" list of list tanpa elemen terakhir list }*

**Contoh-1 : Mengecek kesamaan dua buah list of list:**

**Persoalan :**

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah predikat yang mengecek kesamaan dua buah list of list

KESAMAAN	IsEqS (L1,S2)
<p><b><u>DEFINISI PREDIKAT</u></b></p> <p><b>IsEqS</b> : 2 List of list → boolean</p> <p><i>{IsEqS (S1,S2) true jika S1 identik dengan S2 : semua elemennya sama }</i></p> <p><i>{ Basis : kedua list kosong : → true</i> <i>          salah satu list kosong : → false</i></p> <p><i>Rekurens :</i></p> <p><i>S1</i> <span style="border: 1px solid black; padding: 2px;"><u>L1</u></span> o <span style="border: 1px solid black; padding: 2px;"><i>Tail(S1)</i></span></p> <p><i>S2</i> <span style="border: 1px solid black; padding: 2px;"><u>L2</u></span> o <span style="border: 1px solid black; padding: 2px;"><i>Tail(S2)</i></span></p> <p><i>L1 dan L2 adalah atom : L1=L2 and IsEqS(TailList(S1),TailList(S2))</i> <i>L1 dan L2 adalah list : IsEqS(S1,S2) and IsEqS(TailList(S1),TailList(S2))</i> <i>else : false</i></p> <p><i>}</i></p>	
<p><b><u>REALISASI</u></b></p> <p>IsEqS (S1, S2) :</p> <p>    <u>depend on</u> S1, S2</p> <p>        IsEmpty(S1) <u>and</u> IsEmpty(S2) : <u>true</u></p> <p>        <u>not</u> IsEmpty(S1) <u>and</u> IsEmpty(S2) : <u>false</u></p> <p>        IsEmpty(S1) <u>and</u> <u>not</u> IsEmpty(S2) : <u>false</u></p> <p>        <u>not</u> IsEmpty(S1) <u>and</u> <u>not</u> IsEmpty(S2) :</p> <p>            <u>depend on</u> FirstList(S1), FirstList(S2)</p> <p>                IsAtom(FirstList(S1) <u>and</u> IsAtom(FirstList(S1)) :</p> <p>                    FirstList(S1) = FirstList(S1) <u>and</u></p> <p>                    IsEqS (TailList(S1), TailList(S2))</p> <p>                IsList(FirstList(S1) <u>and</u> IsList(FirstList(S1)) :</p> <p>                    IsEqS(FirstList(S1),FirstList(S2)) <u>and</u></p> <p>                    IsEqS(TailList(S1),TailList(S2))</p> <p>                Else {atom dengan list pasti tidak sama}</p> <p>                    <u>false</u></p>	

**Contoh-2 : Mencek apakah sebuah atom merupakan elemen sebuah list yang elemennya list:**

**Persoalan :**

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah predikat yang mencek keanggotaan sebuah elemen terhadap list of list

KEANGGOTAAN	IsMemberS (A,S)
<p><b><u>DEFINISI PREDIKAT</u></b></p> <p><b>IsMemberS</b> : elemen, <u>List of list</u> → <u>boolean</u></p> <p>{ <i>IsMemberS (A,S) true jika A adalah anggota S</i> }</p> <p>{ <i>Basis : list kosong : → false</i> }</p> <p><i>Rekurens :</i></p> <p style="padding-left: 40px;"> <span style="border: 1px solid black; padding: 2px;">L1</span> ○ <span style="border: 1px solid black; padding: 2px;">Tail(S)</span> </p> <p><i>L1 adalah atom dan A = L1 : true</i></p> <p><i>L1 bukan atom : A anggota L1 or IsMemberS(A, TailList(S))</i></p> <p>}</p>	
<p><b><u>REALISASI</u></b></p> <pre> IsMemberS (A, S) :   depend on S     IsEmpty(S) : false     Not IsEmpty(S) :       depend on FirstList(S)         IsAtom(FirstList(S)) : A = FirstList(S)         IsList(FirstList(S)) : IsMember(A, FirstList(S))           or IsMemberS(A, TailList(S))   { dengan IsMember(A,L) adalah fungsi yang mengirimkan true jika A adalah   elemen list L} </pre>	

**Contoh-3 : Mengecek apakah sebuah List merupakan elemen sebuah list yang elemennya list**

**Persoalan :**

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang mengecek keanggotaan sebuah list terhadap list of list.

KEANGGOTAAN	IsMemberLS (L,S)
<p><b><u>DEFINISI PREDIKAT</u></b></p> <p><b>IsMemberLS</b> : List, List of list → boolean</p> <p>{ <i>IsMemberLS (L,S2) true jika L adalah anggota S</i> }</p> <p>{ <i>Basis : L dan S list kosong : → true</i></p> <p style="padding-left: 40px;"><i>L atau S tidak kosong : false</i></p> <p><i>Rekurens :</i></p> <p style="padding-left: 40px;"> <span style="border: 1px solid black; padding: 2px;"><i>L1</i></span> ○ <span style="border: 1px solid black; padding: 2px;"><i>Tail(S)</i></span> </p> <p><i>L1 adalah atom : IsMemberLS(L,TailList(S))</i></p> <p><i>L1 bukan atom : L1=L : true</i></p> <p style="padding-left: 40px;"><i>L1≠L : IsMemberLS(L,TailList(S))</i></p> <p>}</p>	
<p><b><u>REALISASI</u></b></p> <pre> IsMemberLS (L, S) :   depend on S     IsEmpty(L) and IsEmpty(S) : true     not IsEmpty(L) and IsEmpty(S) : false     IsEmpty(L) and not IsEmpty(S) : false     not IsEmpty(L) and not IsEmpty(S) :       if (IsATOM(FirstList(S))) then IsMemberLS(Taillist(L,S))       else { IsLIST(FirstList(S)) }             if IsEqual(L,FirstList(S)) then true             else IsMemberLS(L,Taillist(S)) </pre> <p>{ dengan IsEqual(L1,L2) adalah fungsi yang mengirimkan true jika list L1 sama dengan list L}</p>	

#### Contoh 4: Menghapus SEBUAH elemen (atom) dari list of list :

##### Persoalan :

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menentukan menghapus sebuah kemunculan atom tertentu yang ada pada sebuah list of list

HAPUS*ELEMEN	Rember*(a,S)
<b>DEFINISI</b>	
<b>Rember*</b> : elemen, <u>List of list</u> → <u>List of list</u> <i>{ Rember* (a,S) menghapus sebuah elemen bernilai a dari semua list S }</i>  <i>{ List kosong tetap menjadi list kosong }</i> <i>{ Basis : list kosong : → ( )</i> <i>Rekurens :</i> $S \quad a \quad L1 \quad o \quad Tail(S)$ <i>L1 adalah atom : L1 = a : TailList(S) tanpa a</i> <i>L1 ≠ a : L1 o (TailList(S) tanpa a)</i> <i>L1 adalah list : (L1 tanpa a) o (TailList(S) tanpa a)</i> <i>}</i>	
<b>REALISASI</b>	
<b>Rember*</b> (a,L) : <pre>if IsEmpty(S) then S else if IsList(FirstList(S)) then KonsLo(Rember*(a,FirstList(S)), Rember*(a,TailList(S))) else { elemen pertama S adalah atom } if FirstElmt(S) = a then Rember*(a,TailList(S)) else KonsLo (FirstElmt(S),Rember*(a,Tail(S))</pre>	

### Contoh 5 : Maksimum dari list of list dengan atom integer:

#### Persoalan :

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menentukan nilai maksimum dari atom-atom yang ada pada sebuah list of list yang tidak kosong.

ELEMEN BERNILAI MAKSIMUM	Max(S)
<b><u>DEFINISI</u></b>	
<p><b>Max</b> <u>List of list</u> tidak kosong <math>\rightarrow</math> <u>integer</u></p> <p><i>{ Max (S) menghasilkan nilai elemen (atom) yang maksimum dari S }</i></p> <p><i>{ Basis : list dengan satu elemen E1 E1 adalah atom : nilai E1 E1 adalah list : Max(E1)</i></p> <p><i>Rekurens :</i></p> <p style="margin-left: 40px;"><i>a</i></p> <p style="margin-left: 40px;"><math>S \quad \boxed{L1} \text{ o } \boxed{\text{Tail}(S)}</math></p> <p><i>L1 adalah atom : Max2(L1,Max(Tail(S))</i> <i>L1 adalah list : Max2 (Max(L1), Max(Tail(S))</i></p> <p><i>}</i></p> <p><i>{Fungsi antara }</i></p> <p><b>Max2</b> 2 <u>integer</u> <math>\rightarrow</math> <u>integer</u></p> <p><i>{Max2(a,b) menghasilkan nilai maksimum a dan b }</i></p>	
<b><u>REALISASI</u></b>	
<pre>Max2 (a,b) :   <u>if</u> a&gt;=b <u>then</u>     a   <u>Else</u>     b</pre> <pre>Max (S) :   <u>if</u> IsOneElmt(S) <u>then</u> {Basis 1 }     <u>if</u> IsAtom(FirstList(S)) <u>then</u>       FirstList(S)     <u>Else</u> { List }       Max(FirstList(S))   <u>Else</u> {Rekurens }     <u>if</u> IsAtom(FirstList(S)) <u>then</u> {First elemen adalah atom }       Max2(FirstList(S),Max(TailList(S))     <u>Else</u> { Firs element adalah List }       Max2(Max(FirstList(S)), Max(TailList(S))</pre>	

## Resume dari analisa rekurens terhadap list

### Rekurens terhadap bilangan integer $n$ : $f(n)$

Basis :  $n = 0$  : ekspresi basis

Rekurens :  $f(\text{prec}(n))$

### Rekurens terhadap list(L) : $f(L)$

Basis kosong :

Basis :  $\text{IsEmpty}(L)$  : ekspresi basis

Rekurens :  $f(\text{Tail}(L))$

Basis satu :

Basis :  $\text{IsOneElmt}(L)$  : ekspresi basis

Rekurens :  $f(\text{Tail}(L))$  { minimal dua elemen }

### Rekurens terhadap list of list (S) : $f(S)$

Basis :  $\text{IsEmpty}(S)$  : ekspresi basis

Rekurens :

depend on  $\text{FirstList}(S)$

$\text{IsAtom}(\text{FirstList}(S))$  :  $g(\text{ekspresi terhadap atom}, f(\text{TailList}(S)))$

$\text{IsList}(\text{FirstList}(S))$ :  $g(\text{ekspresiterhadaplist}, f(\text{Tail}(S)))$

# POHON

## Pendahuluan

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

- mengorganisasi informasi berdasarkan suatu struktur “logik”
- memungkinkan cara akses yang bermacam-macam terhadap suatu elemen

Contoh persoalan yang tepat untuk direpresentasi sebagai pohon:

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

## Definisi rekurens dari pohon:

Sebuah POHON adalah himpunan terbatas tidak kosong, dengan elemen yang dibedakan sebagai berikut :

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

Contoh :

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

## Catatan:

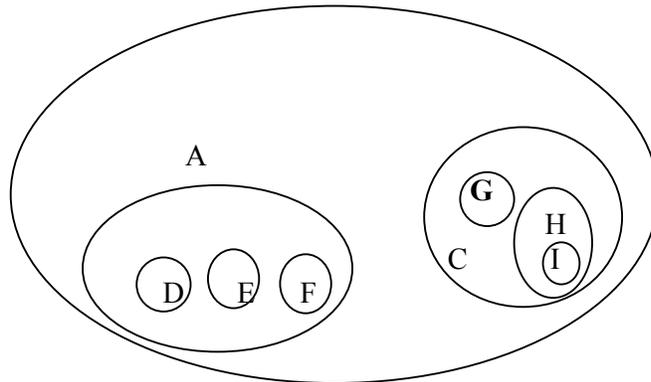
Suffiks (akhiran) n-ary menunjukkan bahwa sub pohon bervariasi semua elemen dari pohon adalah akar dari sub pohon, yang sekaligus menunjukkan pohon tsb

pada definisi di atas, tidak ada urutan sub pohon, namun jika logika dari persoalan mengharuskan suatu strukturasi seperti halnya pada buku, maka dikatakan bahwa pohon berarah

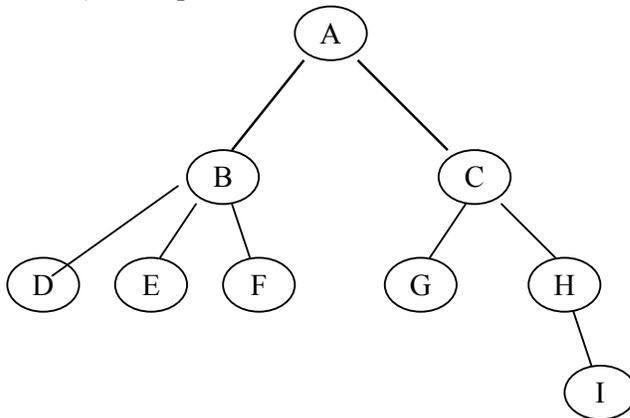
## CARA PENULISAN POHON:

Beberapa ilustrasi representasi berikut ini yang diambil dari [3] merepresentasikan pohon yang sama

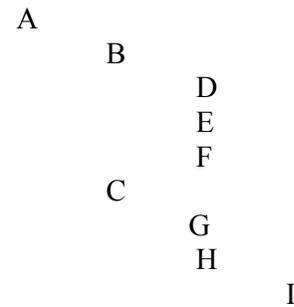
a) Himpunan yang saling melingkupi



b) Graph



c) Indentasi



d) Bentuk linier :

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

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

## BEBERAPA ISTILAH

### HUTAN (*forest*)

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

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

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

**CABANG (*path*):** hubungan antara akar dengan sub pohon

Contoh : pada gambar (B) di atas

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

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

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

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

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

**JALAN (*path*)** adalah suatu urutan tertentu dari CABANG

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

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

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

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

**Catatan:**

Diberikan sebuah pohon biner dengan N elemen. Jika :

- b adalah banyaknya simpul biner
- u adalah banyaknya simpul uner
- d adalah banyaknya daun

Maka akan selalu berlaku:

$$N = b + u + d$$

$$n-1 = 2 b + u$$

sehingga

$$b = d - 1$$

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

## Pohon N-aire

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

### Definisi rekursif

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

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

<b>TYPE POHON-N-AIRE (tidak mungkin kosong)</b>
<b><u>DEFINISI DAN SPESIFIKASI TYPE</u></b> <b>type Elemen</b> : { tergantung type node } <b>type PohonN-ner</b> : $\langle A : \text{Elemen}, PN : \text{PohonN-ner} \rangle$ { notasiPrefix }, atau <b>type PohonN-ner</b> : $\langle PN : \text{PohonN-ner}, A : \text{Elemen} \rangle$ { notasi postfix } <i>{Pohon N-ner terdiri dari Akar yang berupa elemen dan list dari pohon N-aire yang menjadi anaknya List anak mungkin kosong, tapi pohon N-ner tidak pernah kosong, karena minimal mempunyai sebuah elemen sebagai akar pohon}</i>
<b><u>DEFINISI DAN SPESIFIKASI SELEKTOR</u></b> <b>Akar</b> : PohonN-ner tidak kosong $\rightarrow$ Elemen <i>{ Akar(P) adalah Akar dari P. Jika P adalah (A,PN) = Akar(P) adalah A }</i> <b>Anak</b> : PohonN-ner tidak kosong $\rightarrow$ list of PohonN-ner <i>{ Anak(P) adalah list of pohon N-ner yang merupakan anak-anak (sub phon) dari P. Jika P adalah (A, PN) = Anak (P) adalah PN }</i>
<b><u>DEFINISI DAN SPESIFIKASI KONSTRUKTOR</u></b> <i>{ Perhatikanlah bahwa konstruktor pohon N-ner dengan basis pohon kosong dituliskan sebagai</i> <i>a. Prefix : (A,P,N)</i> <i>b. Posfix : (PN,A) }</i>
<b><u>DEFINISI DAN SPESIFIKASI PREDIKAT</u></b> <b>IsTreeNEmpty</b> : PohonN-ner $\rightarrow$ <u>boolean</u> <i>{IsTreeNEmpty(PN) true jika PN kosong : () }</i> <b>IsOneElmt</b> : PohonN-ner $\rightarrow$ <u>boolean</u> <i>{IsOneElmt(PN) true jika PN hanya terdiri dari Akar }</i>

### DEFINISI DAN SPESIFIKASI PREDIKAT LAIN

**NbNElmt** : PohonN-ner  $\rightarrow$  integer  $\geq 0$

*{NbNElmt(P) memberikan banyaknya node dari pohon P :*

*Basis 1: NbNElmt ((A)) = 1*

*Rekurens : NbNElmt ((A,PN)) = 1 + NbELmt(PN) }*

**NbNDAun** : PohonN-ner  $\rightarrow$  integer  $\geq 0$

*{NbNDAun (P) memberikan banyaknya daun dari pohon P :*

*Basis 1: NbNDAun (A) = 1*

*Rekurens : NbNDAun ((A,PN)) = NbNDAun(PN)*

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

## Pohon Biner

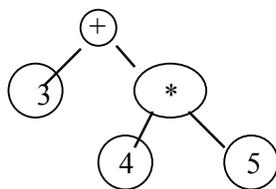
### Definisi :

sebuah pohon biner adalah himpunan terbatas yang

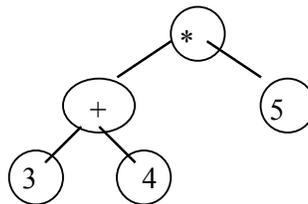
- mungkin kosong, atau
- terdiri dari sebuah simpul yang disebut akar dan dua buah himpunan lain yang *disjoint* yang merupakan **pohon biner**, yang disebut sebagai sub pohon kiri dan sub pohon kanan dari pohon biner tersebut

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

Contoh pohon ekspresi aritmatika

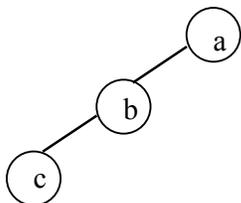


$3 + (4 * 5)$

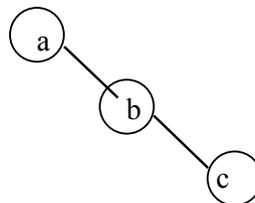


$(3 + 4) * 5$

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



Pohon biner condong kiri

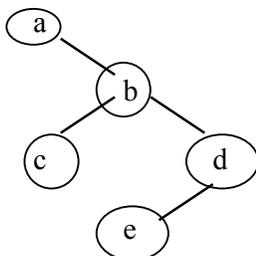


Pohon biner condong kanan

Sub pohon ditunjukkan dengan penulisan ( )

Notasi prefix :

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



### Definisi rekursif pohon biner basis-0

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

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

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

#### **DEFINISI DAN SPESIFIKASI TYPE**

**type Elemen** : { tergantung type node }

**type PohonBiner** :  $\langle L : \text{PohonBiner}, A : \text{Elemen}, R : \text{PohonBiner} \rangle$  {notasi Infix}, atau

**type PohonBiner** :  $\langle A : \text{Elemen}, L : \text{PohonBiner}, R : \text{PohonBiner} \rangle$  {notasi prefix}, atau

**type PohonBiner** :  $\langle L : \text{PohonBiner}, R : \text{PohonBiner}, A : \text{Elemen} \rangle$  {notasi postfix }

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

#### **DEFINISI DAN SPESIFIKASI SELEKTOR**

**Akar** : PohonBiner tidak kosong  $\rightarrow$  Elemen

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

**Left** : PohonBiner tidak kosong  $\rightarrow$  PohonBiner

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

**Right** : PohonBiner tidak kosong  $\rightarrow$  PohonBiner

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

#### **DEFINISI DAN SPESIFIKASI KONSTRUKTOR**

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

a. *Infix* : /L A R\

b. *Prefix* : /A L R\

c. *Posfix* : /L R A\ }

#### **DEFINISI DAN SPESIFIKASI PREDIKAT**

**IsEmpty** : PohonBiner  $\rightarrow$  boolean

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

#### **DEFINISI DAN SPESIFIKASI PREDIKAT LAIN**

**NbElmt** : PohonBiner  $\rightarrow$  integer  $\geq 0$

*{NbElmt(P) memberikan Banyaknya elemen dari pohon P :*

*Basis : NbElmt ( $\wedge \backslash$ ) = 0*

*Rekurens : NbElmt ( $\wedge L,A,R \backslash$ ) = NbElmt(L) + 1 + NbELmt(R) }*

**NbDaun** : PohonBiner  $\rightarrow$  integer  $\geq 0$

*{ definisi : Pohon kosong berdaun 0 }*

*{NbDaun (P) memberikan Banyaknya daun dari pohon P :*

*Basis-1 : NbDaun ( $\wedge \backslash$ ) = 0*

*Rekurens :*

**NbDaun1** (P)

**RepPrefix**: PohonBiner  $\rightarrow$  list of element

*{RepPrefix (P) memberikan representasi linier (dalam bentuk list), dengan urutan elemen list sesuai dengan urutan penulisan pohon secara prefix :*

*Basis : RepPrefix ( $\wedge \backslash$ ) = []*

*Rekurens : RepPrefix ( $\wedge L,A,R \backslash$ ) = [A] o RepPrefix(L) o RepPrefix (R) }*

## **REALISASI**

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

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

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

### Definisi rekursif pohon biner basis-1

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

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

#### **DEFINISI DAN SPESIFIKASI TYPE**

**type** Elemen : { tergantung type node }

**typ}** PohonBiner :  $\langle L : \text{PohonBiner}, A : \text{Elemen}, R : \text{PohonBiner} \rangle$  {notasi Infix}, atau

**type** PohonBiner :  $\langle A : \text{Elemen}, L : \text{PohonBiner}, R : \text{PohonBiner} \rangle$  {notasi prefix }, atau

**type** PohonBiner :  $\langle L : \text{PohonBiner}, R : \text{PohonBiner}, A : \text{Elemen} \rangle$  {notasi postfix }

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

#### **DEFINISI DAN SPESIFIKASI SELEKTOR**

**Akar** : PohonBiner tidak kosong  $\rightarrow$  Elemen

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

**Left** : PohonBiner tidak kosong  $\rightarrow$  PohonBiner

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

**Right** : PohonBiner tidak kosong  $\rightarrow$  PohonBiner

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

#### **DEFINISI DAN SPESIFIKASI KONSTRUKTOR**

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

a. *Infix* : //L A R\

b. *Prefix* : //A L R\

c. *Posfix* : //L R A\

*atau bahkan notasi lain yang dipilih}*

#### **DEFINISI DAN SPESIFIKASI PREDIKAT**

**IsEmpty** : PohonBiner  $\rightarrow$  boolean

*{IsEmpty(P) true jika P kosong : (// \ \) }*

**IsOneElmt** : PohonBiner  $\rightarrow$  boolean

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

**IsUnerLeft** : PohonBiner  $\rightarrow$  boolean

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

**IsUnerRight** : PohonBiner  $\rightarrow$  boolean

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

**IsBiner** : PohonBiner tidak kosong  $\rightarrow$  boolean

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

**IsExistLeft** : PohonBiner tidak kosong  $\rightarrow$  boolean

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

**IsExistRight** : PohonBiner tidak kosong  $\rightarrow$  boolean

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

### DEFINISI DAN SPESIFIKASI PREDIKAT LAIN

**NbElmt** : PohonBiner  $\rightarrow$  integer  $\geq 0$

*{NbElmt(P) memberikan Banyaknya elemen dari pohon P :*

*Basis : NbElmt ( $//A\backslash$ ) = 1*

*Rekurens : NbElmt ( $//L,A,R\backslash$ ) = NbElmt(L) + 1 + NbELmt(R)*

*NbElmt ( $//L,A,\backslash$ ) = NbElmt(L) + 1*

*NbElmt ( $//A,R\backslash$ ) = 1 + NbELmt(R) }*

**NbDaun1** : PohonBiner  $\rightarrow$  integer  $\geq 1$

*{ Prekondisi : Pohon P tidak kosong }*

*{NbDaun (P) memberikan Banyaknya daun dari pohon P :*

*Basis : NbDaun1 ( $//A\backslash$ ) = 1*

*Rekurens : NbDaun1 ( $//L,A,R\backslash$ ) = NbDaun1 (L) + NbDaun1(R)*

*NbDaun1 ( $//L,A,\backslash$ ) = NbDaun1 (L)*

*NbDaun1 ( $//A,R\backslash$ ) = NbDaun1 (R)*

**RepPrefix** : PohonBiner  $\rightarrow$  list of element

*{RepPrefix (P) memberikan representasi linier (dalam bentuk list), dengan urutan elemen list sesuai dengan urutan penulisan pohon secara prefix :*

*Basis : RepPrefix ( $//A\backslash$ ) = [A]*

*Rekurens : RepPrefix ( $//L,A,R\backslash$ ) = [A] o RepPrefix(L) o RepPrefix (R)*

*RepPrefix ( $//L,A,\backslash$ ) = [A] o RepPrefix(L)*

*RepPrefix ( $//A,R\backslash$ ) = [A] o RepPrefix (R)*

*}*

### REALISASI

**NbElmt** (P) : {P tidak kosong }

if IsOneElmt(P) then 1

else depend on P

IsBiner (P) : NbElmt (Left (P) + 1 + NbElmt (Right (P)

IsUnerLeft (P) : NbElmt (Left (P) + 1

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

```
NbDaun (P) :
```

```
  if 1Element?(P) then {Basis}
```

```
    1
```

```
  else {Rekurens }
```

```
    depend on P
```

```
      IsBiner(P) : NbDaun1(Left(P)) + NbDaun1(Right(P))
```

```
      IsUnerLeft(P) : NbDaun1(Left(P))
```

```
      IsUnerRight(P) : NbDaun1(Right(P))
```

```
RepPrefix (P) :
```

```
  if 1Elmt?(P) then [Akar(P)]
```

```
  else depend on P
```

```
    IsBiner(P) : KonsoL(KonsoL(Akar(P), RepPrefix(Left(P))),  
                       RepPrefix(Right(P)))
```

```
    IsUnerLeft(P) : KonsoL(Akar(P), RepPrefix(Left(P)))
```

```
    IsUnerRight(P) : KonsoL(Akar(P), RepPrefix(Right(P)))
```

**Latihan soal :**

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

Buatlah realisasi dari spesifikasi fungsional terhadap pohon biner sebagai berikut

**DEFINISI DAN SPESIFIKASI PREDIKAT LAIN**

IsMember : PohonBiner, elemen  $\rightarrow$  boolean

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

*{ fungsi lain }*

IsSkewLeft: PohonBiner  $\rightarrow$  boolean

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

IsSkewRight : PohonBiner  $\rightarrow$  boolean

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

LevelOfX: PohonBiner, elemen  $\rightarrow$  integer

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

*{ Operasi lain }*

AddDaunTerkiri : PohonBiner, elemen  $\rightarrow$  PohonBiner

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

AddDaun : PohonBiner tidak kosong, node,node, boolean  $\rightarrow$  PohonBiner

*{ AddDaun (P,X,Y,Kiri) : P bertambah simpulnya, dengan Y sebagai anak kiri X (jika Kiri), atau sebagai anak Kanan X (jika not Kiri)*

*{ Prekondisi : X adalah salah satu daun Pohon Biner P }*

DelDaunTerkiri: PohonBiner tidak kosong,  $\rightarrow$  <PohonBiner,elemen >

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

DelDaun : PohonBiner tidak kosong, elemen  $\rightarrow$  PohonBiner

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

MakeListDaun : PohonBiner  $\rightarrow$ list Of Node

*{MakeListDaun(P) : }*

*{Jika P adalah pohon kosong, maka menghasilkan list kosong.*

*{Jika P bukan pohon kosong: menghasilkan list yang elemennya adalah semua daun pohon P}*

MakeListPreOrder : PohonBiner)  $\rightarrow$ list Of Node

*{MakeListPreOrder(P) : }*

*{Jika P adalah pohon kosong, maka menghasilkan list kosong. }*

*{Jika P bukan pohon kosong: menghasilkan list yang elemennya adalah semua node pohon P dengan urutan Preorder}*

MakeListPostOrder : PohonBiner  $\rightarrow$ list Of Node

*{MakeListPostOrder(P) : }*

*{Jika P adalah pohon kosong, maka menghasilkan list kosong. }*

*{Jika P bukan pohon kosong: menghasilkan list yang elemennya adalah semua node pohon P dengan urutan PostOrder}*

MakeListInOrder : PohonBiner  $\rightarrow$ list Of Node

*{MakeListInOrder(P) : }*

*{Jika P adalah pohon kosong, maka menghasilkan list kosong. }*

*{Jika P bukan pohon kosong: menghasilkan list yang elemennya adalah semua node pohon P dengan urutan InOrder}*

MakeListLevel : PohonBiner, integer  $\rightarrow$ list Of Node

*{MakeListLevel(P,N) : }*

*{Jika P adalah pohon kosong, maka menghasilkan list kosong. }*

*{Jika P bukan pohon kosong: menghasilkan list yang elemennya adalah semua node pohon P yang levelnya=N}*

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

## Binary Search Tree

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

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

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

**BSearchX** : BinSearchTree, elemen  $\rightarrow$  boolean

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

**AddX**: BinSearchTree, elemen  $\rightarrow$  PohonBiner

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

**MakeBinSearchTree**: list of elemen  $\rightarrow$  PohonBiner

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

**DelBtree**: BinSearchTree tidak kosong, elemen  $\rightarrow$  PohonBiner

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

## **Pohon Seimbang (*balanced tree*)**

Definisi Pohon seimbang:

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

Buatlah realisasi dari definisi dan spesifikasi sebagai berikut sebagai latihan

BuildBalanceTree: list of node, integer  $\rightarrow$  BinBalTree

*{ Meghasilkan sebuah balance tree dengan n node, nilai setiap node yang berasal dari list }*

## EKSPRESI LAMBDA

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

### Fungsi sebagai domain dari fungsi (parameter)

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

Selain “penangguhan” fungsi pada saat eksekusi, konsep ekspresi lambda dibutuhkan untuk menggeneralisasi fungsi.

Contohnya :

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

Beberapa persoalan matematik, menghasilkan fungsi sebagai hasil dari komputasi fungsi. Misalnya derivasi suatu fungsi polinomial akan menghasilkan suatu fungsi polinomial berderajat satu kurang dari fungsi asal. Untuk memenuhi kebutuhan ini, notasi fungsional diperluas sehingga range dari sebuah fungsi akan menghasilkan fungsi.

Translasi konsep ini ke bahasa pemrograman membutuhkan mekanisme yang sangat spesifik bahasa. Bahkan hampir tidak ada bahasa fungsional yang mampu menterjemahkan konsep ini secara satu ke satu tanpa melalui mekanisme yang tersedia.

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

Deskripsi persoalan : untuk suatu kebutuhan melakukan penjumlahan deret yang “mirip”, mula-mula didefinisikan tiga buah fungsi yang terpisah. Semua jumlah disebut sebagai “Sigma” namun nilai yang akan dijumlahkan yang merupakan fungsi I anggota interval berbeda-beda.

Kemudian, karena kemiripan rumusnya, ingin dibentuk sebuah fungsi yang dapat mewakili ketiga fungsi tersebut. Tahapan-tahapannya diberikan lewat contoh sebagai berikut:

Perhatikan tiga buah fungsi SigI, SigI3 dan SP8 sebagai berikut :

Suatu interval akan didefinisikan secara rekurens sebagai berikut :

**Basis:** interval kosong artinya nilai  $a > b$ , yaitu tidak ada lagi daerah yang merupakan definisi interval

**Rekurens :** akan diberikan ilustrasi analisa rekurens terhadap interval dianalogikan terhadap list sebagai berikut :



### DEFINISI DAN SPESIFIKASI

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

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

### REALISASI

```

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

### DEFINISI DAN SPESIFIKASI

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

**SigI3:** 2 integer → integer

*{SigI3(a,b) adalah fungsi untuk menghitung Sigma(i<sup>3</sup>) untuk nilai i pada interval a dan b:  $a^3 + (a+1)^3 + (a+1+1)^3 + \dots + b^3$ , atau 0 jika interval "kosong" }*

### REALISASI

```

SigI3(a,b) :
    if a>b then {Basis-0}
        0
    else {Rekurens}
        a3 + SigI3(a+1,b)
    
```

### DEFINISI DAN SPESIFIKASI

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

**SP8** : integer → real

{SP8(a,b) adalah fungsi untuk menghitung deret konvergen ke  $\pi/8$  pada interval a dan b atau 0 jika interval "kosong". Rumus :

$1/(1*3) + 1/(5*7) + 1/(9*11) + \dots$  }

### **REALISASI**

**SP8** (a,b) :

if a>b then {Basis-0}  
0

else {Rekurens}  
(1/((a)\*(a+2)) + SP8(a+4,b)

Definisikan fungsi-fungsi berikut:

### **DEFINISI DAN SPESIFIKASI**

**Id** : integer → integer

{ Id(i) mengirimkan nilai i

Id(i) : i

**P1** : integer → integer

{ P1(i) mengirimkan nilai i+1

P1(i) : i+1

**P4** : integer → integer

{ P4(i) mengirimkan nilai i+4

P4(i) : i+4

**Cube** : integer → integer

{ Cube(i) mengirimkan nilai  $i^3$

Id(i) :  $i^3$

**T** : integer → real

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

T(i) :  $1/((i+1)*(i+3))$

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

Definisikan "Sigma" dari deret :

b

$\Sigma_{n=a} f(n) = f(a) + \dots + f(b)$

n=a

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

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

<p><b><u>DEFINISI</u></b></p> <p><b>type numerik</b> : union dari type <u>integer</u> dan <u>real</u></p> <p><b>Sigma</b> : <u>integer</u>, <u>integer</u>,(<u>integer</u> → numerik),(<u>integer</u> → numerik) → numerik</p> <p><i>{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 }</i></p>
<p><b><u>REALISASI</u></b></p> <pre> Sigma (a,b,f,s) :     <u>if</u> a&gt;b <u>then</u> {Basis-0}         0     <u>else</u> {Rekurens}         f(a) + Sigma(s(a),b,f,s)                 </pre>

Maka :

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

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

**Caranya adalah dengan ekspresi LAMBDA**

Perhatikan ekspresi sbb :

```

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

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

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

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

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

**Id** :  $\lambda x.x$   
**Cube** :  $\lambda x.x^3$   
**P1** :  $\lambda x.x+1$   
**P4** :  $\lambda x.x+4$

Aplikasi terhadap ekspresi lambda :

Tuliskan

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

Evaluasinya akan sama.

Maka fungsi Sigma dapat dituliskan sbb :

Untuk SigI :  $\text{Sigma}(a,b, \lambda x.x, \lambda x.x+1)$

Untuk SigI3 :  $\text{Sigma}(a,b, \lambda x.x^3, \lambda x.x+1)$

Untuk SP8 :  $\text{Sigma}(a,b, \lambda x.1/((x+1)+(x+3)), x.x+4)$

Hasil evaluasi sesuai dengan type hasil ekspresi

Ekspresi lambda dengan parameter banyak dituliskan sebagai:

$\lambda x,y.x+y$

adalah fungsi untuk menjumlahkan nilai x dan y.

Kedua ekspresi berikut adalah ekivalen :

$\lambda x,y. x+y$

$\lambda x. \lambda y. x+y$

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

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

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

$2+3=5$

Perhatikan bahwa  $\lambda x,y.x+y$  dapat diaplikasi dengan satu parameter saja.

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

$\lambda y. 2+y$

$(\lambda x. \lambda y. x+y)$  (2)

menambahkan dua ke nilai y

### Fungsi Sebagai Hasil Dari Evaluasi (Range)

Perhatikan bahwa derivasi dari  $f(x) = x^3$  adalah sebuah fungsi  $f'(x) = 3x^2$ , sedangkan nilai derivasi  $f'(x)$  untuk  $x=2$  adalah suatu nilai numerik.

Maka derivasi suatu fungsi pada suatu titik dapat didefinisikan sbb:

Derivasi (real  $\rightarrow$  real), real  $\rightarrow$  real  $\rightarrow$  real

DerivTitikX (real  $\rightarrow$  real), real, real  $\rightarrow$  real

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

derivasi untuk  $f(y) = y^3$  dengan notasi lambda adalah

$\lambda x. ( (\lambda y. y^3) (x+dx) - (\lambda y. y^3) (x) ) / dx$

Sehingga DerivTitikX untuk  $f(y) = y^3$  dan nilai  $dx=0.005$  dan NilaiX = 5 dituliskan sebagai aplikasi dari DerivTitikX dengan parameter

$(\lambda y.y^3, 0.005)$  (5)

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

### **DEFINISI**

**Derivasi** : (real  $\rightarrow$  real), real  $\rightarrow$  (real  $\rightarrow$  real)

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



Ekspresi lambda di atas berarti : tambahkan n pada f

Dengan static binding (pada saat definisi) :

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

Dengan dynamic binding (pada saat aplikasi) :

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

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

## Studi kasus lewat contoh

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

#### Persoalan :

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

Contoh :

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

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

OFFSETLIST	OffsetList(List,Offset )
<b>DEFINISI DAN SPESIFIKASI</b>	
<b>OffsetList</b> : <u>list of integer tidak kosong</u> , Offset → <u>list of integer</u> <i>{OffsetList (Li,Offset), dengan Li adalah list integer dan Offset adalah sebuah fungsi dengan definisi melakukan offset. OffsetList menghasilkan sebuah list integer dengan elemen yang sudah dioffset}</i>	
<b>REALISASIP</b>	
<pre>OffsetList(Li, Offset) :   Konso (Offset (FirstElmt (Li), OffsetList (Tail (Li), Offset) )</pre>	
<b>BEBERAPA CONTOH OFFSET</b>	
<i>{f adalah Plus 2 }</i> <b>Offset</b> ≡ <b>Plus2(i)</b> : $i + 2$ <i>{f adalah Minus 1 }</i> <b>Offset</b> ≡ <b>Minus1(i)</b> : $i - 1$	

*{f} adalah ekspresi kondisional }*

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

### APLIKASI

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

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

### Persoalan :

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

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

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

<b>FILTERLIST</b>	<b>FilterList(List,f)</b>
<b><u>DEFINISI DAN SPESIFIKASI</u></b>	
<b>FilterList</b> : <u>list of integer tidak kosong</u> , $f \rightarrow$ <u>list of integer</u> <i>{FilterList (Li, f), dengan Li adalah list integer dan f adalah sebuah predikat dengan definisi f(i) menghasilkan sebuah list integer dengan elemen yang memenuhi Predikat f}</i>	
<b><u>REALISASI</u></b>	
<pre>filterList(Li, f) :     if not f(FirstElmt(Li))     then filterList(Li, f)     else Konso(FirstElmt(Li), filterList(Li, f) )</pre>	
<b><u>BEBERAPA CONTOH FUNGSI F</u></b>	
<i>{ filter adalah integer positif : IsPos? (i) benar jika i positif }</i> $f \equiv \text{IsPos}(i) : i > 0$ ) <i>{ filter adalah integer positif : IsNeg? (i) benar jika i negatif }</i> $f \equiv \text{IsNeg}(i) : i < 0$ ) <i>{ filter adalah: Kabisat?(i) : bilangan kelipatan 4 tapi bukan kelipatan 100 }</i> $f \equiv \text{Kabisat}(i) ; . (i \bmod 4 = 0) \text{ and } (i \bmod 100 \neq 0)$ )	
<b><u>APLIKASI</u></b>	
⇒ FilterList([1,3,6,0,-9,45], IsPos)	

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

### **APLIKASI**

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

### **Contoh-3 : FilterRekList : Ekspresi lambda rekursif**

#### **Persoalan :**

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

<b>FilterRekLIST</b>	<b>FilterRekList(List,f)</b>
<b><u>DEFINISI DAN SPESIFIKASI</u></b>	
<p><b>FilterRekList</b> : <u>list of list integer tidak kosong</u> , fungsi → <u>list of integer</u></p> <p>{ <i>FilterRekList (Si, f)</i>, dengan <i>Si</i> adalah <i>list of list integer</i> dan <i>f</i> adalah sebuah fungsi dengan definisi diberikan suatu <i>list of list integer</i> menghasilkan <i>list integer</i> yang menjadi anggota list yang dihasilkan oleh <i>f</i>}</p>	
<b><u>REALISASI</u></b>	
<p><b>FilterRekList</b>(Si, f) :</p> <pre>Konso (f(FirstList(Si), FilterRekList(TailList(Si),f))</pre>	
<b><u>BEBERAPA CONTOH F</u></b>	
<p>{<i>Contoh ekspresi f(S)</i> adalah sebuah fungsi yang menerima sebuah <i>list of list integer</i> dan menghasilkan <i>list integer</i> berasal dari atom-atom list }</p> <pre>f(S) ≡ LINEAR (S)       if Isatom? (FirstList(S))       then Konso(FirstList(S), LINEAR(TailList(S))       else { bukan atom, harus dilinearkan }            Konso(LINEAR(FirstList(S),LINEAR(TailList(S)))</pre> <p>{<i>Contoh ekspresi FilterRek(S)</i> adalah sebuah fungsi yang menerima sebuah <i>list of list integer</i> dan menghasilkan <i>list integer</i> berasal dari atom-atom list, hanya jika atomnya positif }</p> <pre>f(S) ≡       if Isatom? (FirstList(S))       then if FirstList(S) &gt; 0 then FirstList(S)            else []       else { bukan atom, harus dilinearkan }            Konso(FilterRek(FirstList(S), FilterRek(TailList(S)))</pre>	
<b><u>APLIKASI</u></b>	
⇒	

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