

DRAFT DIKTAT KULIAH

DASAR PEMROGRAMAN

Bagian: Pemrograman Fungsional

Oleh :
Inggriani Liem



Kelompok Keahlian Data & Rekayasa Perangkat Lunak
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Revisi Februari 2014

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). Pada kurikulum 2008 dan 2013, diktat ini diterbitkan untuk menunjang kuliah 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 mengharap agar kesalahan dapat disampaikan agar dapat diperbaiki pada versi berikutnya.

DAFTAR ISI

KATA PENGANTAR	2
DAFTAR ISI.....	3
PENDAHULUAN	4
Paradigma Pemrograman	4
Bahasa Pemrograman.....	6
Belajar Memrogram Tidak Sama dengan Belajar Bahasa Pemrograman.....	7
Tujuan Kuliah Pemrograman Fungsional	8
Ikhtisar Diktat.....	9
Daftar Pustaka	10
NOTASI FUNGSIONAL	11
EKSPRESI DASAR.....	15
Jenis-Jenis Ekspresi Dasar	15
Evaluasi Fungsi	22
Ekspresi Bernama dan Nama Antara	24
EKSPRESI KONDISIONAL.....	26
Notasi Ekspresi Kondisional	26
Evaluasi Ekspresi Kondisional.....	27
Operator Boolean AND THEN dan OR ELSE	27
Latihan Soal	33
TYPE BENTUKAN.....	35
Pendefinisian Type Bentukan	35
Fungsi dengan Range Type Bentukan Tanpa Nama	43
Latihan Soal	45
EKSPRESI REKURSIF.....	46
Type Rekursif.....	47
Fungsi Rekursif	49
Ekspresi Rekursif terhadap Bilangan Bulat	52
KOLEKSI OBJEK	56
LIST	58
List dengan Elemen Sederhana	59
List of Character (Teks)	70
List of Integer	74
Himpunan (Set)	80
List of List	88
Resume Analisa Rekurens.....	95
POHON	96
Pendahuluan	96
Beberapa Istilah.....	97
Pohon N-aire	99
Pohon Biner.....	100
Latihan Soal	105
Binary Search Tree.....	107
Pohon Seimbang (<i>Balanced Tree</i>)	107
EKSPRESI LAMBDA	108
Fungsi sebagai Domain dari Fungsi (Parameter).....	108
Fungsi sebagai Hasil dari Evaluasi (Range).....	112
Studi Kasus	114

PENDAHULUAN

Paradigma Pemrograman

Komputer digunakan sebagai alat bantu penyelesaian suatu persoalan. Masalahnya adalah 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-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 priotas 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" memrogram atau memrogram 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, tidak mungkin semua paradigma disatukan dalam sebuah mata kuliah. Di Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika ITB, mahasiswa belajar memrogram diawali 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 memrogram 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, salah satu bagian dari buku ini akan mencakup aspek prosedural dalam paradigma pemrograman fungsional.

Beberapa paradigma dalam pemrograman:

1. Paradigma Pemrograman Prosedural (Imperatif)

Paradigma ini didasari oleh konsep mesin Von Newmann (*stored program concept*), yaitu 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:

$$\text{Algoritma} + \text{Struktur Data} = \text{Program}$$

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

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 **transformasional**. 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 memorisasi 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/Lojik

Paradigma ini didasari oleh pendefinisian **relasi** antar individu yang dinyatakan sebagai **predikat orde pertama**. Sebuah program lojik 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.

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 hierarki, 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, yaitu 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.

5. Paradigma Pemrograman 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".

Bahasa Pemrograman

Berbicara mengenai bahasa pemrograman, ada banyak sekali bahasa pemrograman, mulai dari bahasa tingkat rendah (bahasa mesin dalam biner), bahasa asembler (dalam kode mnemonik), bahasa tingkat tinggi, sampai bahasa generasi keempat (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, Haskell
3. Deklaratif: Prolog
4. *Object oriented* murni: Smalltalk, Eifel, 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 Memrogram Tidak Sama dengan Belajar Bahasa Pemrograman

Belajar memrogram 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 memrogram lebih bersifat pemahaman persoalan, analisis, sintesis. Sedangkan 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 memrogram lebih bersifat keterampilan dari pada analisis dan sintesa.

Belajar memrogram 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 memrogram adalah proses yang memerlukan kepakaran, proses *coding* lebih merupakan proses semi otomatis dengan aturan pengkodean. Proses memrogram memang berakhir secara konkret 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譬喻为 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 sekvens/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:

Buku I – Notasi Fungsional difokuskan kepada pemrograman fungsional dengan kasus-kasus tipikal serta primitif yang berguna untuk memrogram 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 detil)

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 bagian 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 pemakaian 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 dituliskan 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, diusahakan agar instruksi yang dipakai hanyalah instruksi standar.

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 memrogram 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 :

PROGRAM PLUSAB

{ Membaca dua buah nilai a dan b integer, menghitung jumlahnya dan menuliskan hasilnya }

Kamus :

a, b : integer

Algoritma :

input (a, b)
output (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 konteks 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*).

APLIKASI

⇒ 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 strukturalisasi 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, dapat digunakan kemudian.

Konstruksi program fungsional : definisi – spesifikasi – realisasi – aplikasi

Tahapan	Deskripsi
Definisi	Kata kuncinya adalah memberikan identitas fungsi , yaitu menentukan nama, domain dan range . Contoh: untuk memangkatkan sebuah bilangan integer dengan tiga, didefinisikan nama Pangkat3, dengan domain integer dan range adalah integer. Dituliskan Pangkat3 : integer → integer
Spesifikasi	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.
Realisasi	Kata kuncinya adalah menentukan " bagaimana " fungsi melakukan komputasi, yaitu mengasosiasikan pada nama fungsi, sebuah ekspresi fungsional 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.
Aplikasi	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 yang merupakan deskripsi sangat ringkas dari fungsi berikut 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 bagian ini (definisi dan spesifikasi) tidak dipisahkan satu sama lain karena sangat erat kaitannya.

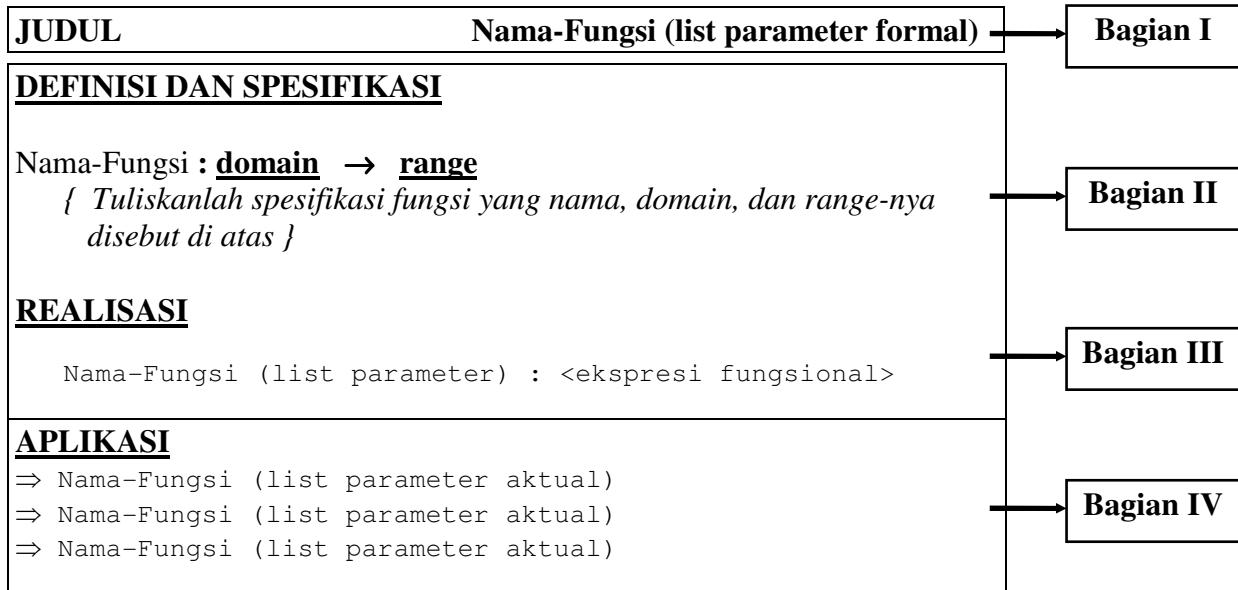
Bagian ketiga berisi realisasi fungsi, yaitu **ekspresi fungsional** yang ditulis untuk mencapai spesifikasi yang dimaksudkan. Sebuah definisi dan spesifikasi yang sama dapat direalisasikan dalam beberapa ekspresi. Perhatikanlah pula bahwa dalam realisasi fungsi, nama fungsi dituliskan berikut ekspresinya. 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 konteks 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 ketiaadaan 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, 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	$*, /, +, -$ <u>mod</u> <u>div</u>	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	$<, >, =, \leq, \geq, \neq$	Berlaku untuk operan numerik atau karakter, hasilnya adalah boolean
Operator boolean	<u>and</u> , <u>or</u> , <u>not</u>	Berlaku untuk operan boolean, sesuai dengan definisi <u>and</u> , <u>or</u> , dan <u>not</u> pada aljabar boolean

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

Jenis-Jenis Ekspresi Dasar

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**. Sebuah ekspresi boolean yang 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.

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

PANGKAT3 (versi-1)	FX3(x)
DEFINISI DAN SPESIFIKASI	
FX3 : <u>integer</u> → <u>integer</u> <i>{ FX3 (x) menghitung pangkat tiga dari x, sebuah bilangan integer }</i>	
REALISASI	
FX3 (x) : x * x * x	
APLIKASI	
$\Rightarrow \text{FX3}(1)$ $\Rightarrow \text{FX3}(8)$ $\Rightarrow \text{FX3}(-1)$	

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.

PANGKAT3 (versi 2)	FX3(x)
DEFINISI DAN SPESIFIKASI	
FX3 : <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>	
FX2 : <u>integer</u> → <u>integer</u> <i>{ FX2 (x) menghitung pangkat dua dari x, sebuah bilangan integer }</i>	
REALISASI	
FX3 (x) : x * FX2(x)	
APLIKASI	
$\Rightarrow \text{FX3}(8)$ $\Rightarrow \text{FX3}(1)$ $\Rightarrow \text{FX3}(-1)$	

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.

MAKSIMUM 3 BILANGAN INTEGER	MAX3(a,b,c)
DEFINISI DAN SPESIFIKASI	
MAX3 : 3 <u>integer</u> → <u>integer</u>	{ MAX3 (a,b,c) menentukan nilai maksimum dari 3 bilangan integer a,b, dan c }
MAX2 : 2 <u>integer</u> → <u>integer</u>	{ MAX2 (a,b) menentukan nilai maksimum dua buah bilangan integer a dan b }
REALISASI	
{ Realisasi MAX2(a,b) pada contoh yang lain }	
MAX3 (a,b,c) : MAX2(MAX2(a,b),c)	
APLIKASI	
⇒ MAX3(8,1,2)	
⇒ MAX3(8,10,-2)	
⇒ MAX3(0,0,0)	

Catatan :

Beberapa fungsi seperti MAX3 yang mempunyai definisi dan arti semantik yang sama direalisasi dengan nama yang sama. Pada dunia nyata, akan terjadi konflik nama, karena sebuah nama hanya boleh muncul secara unik dalam suatu “univers” (lingkungan tertentu). Jangan lupa melakukan penamaan berbeda jika melakukan translasi ke bahasa pemrograman nyata, dianjurkan dengan mempertahankan nama asli, kemudian identifikasi versi misalnya MAX3v1, MAX3v2, dan seterusnya.

Contoh-4 Ekspresi numerik: Mean Olympique (MO)

Persoalan :

Definisikan sebuah fungsi yang menerima 4 bilangan bulat positif, menghasilkan harga rata-rata dari dua di antara empat buah bilangan tersebut, dengan mengabaikan nilai terbesar dan nilai terkecil.

Contoh : $10,8,12,14 = 11$; $12,12,12,12 = 12$

1. Definisi : menentukan nama fungsi, domain dan range

MO : 4 bilangan bulat $> 0 \rightarrow \text{real}$ { Nama fungsi : MO }
{ Domain : 4 buah bilangan bulat }
{ Range : bilangan riil }

Contoh : MO (10,8,12,14) = 11

2. Realisasi dari definisi tersebut dapat dilakukan, misalnya berdasarkan tiga buah ide sebagai berikut :

a. SORT: urutkan mengecil, buang terbesar + terkecil, hitung rata-rata dari 2 sisanya.

- b. Buang maksimum dan minimum: buang tebesar, buang terkecil, hitung rata-rata dari yang tersisa.
- c. Kalkulasi: $MO = \text{jumlah ke empat angka, dikurangi dengan terbesar, dikurangi dengan terkecil.}$

Ide pertama (SORT) dan kedua (buang nilai, kalkulasi) adalah aksional dan harus dilakukan secara “sekuensial”, berurutan. Untuk memecahkan persoalan ini dibutuhkan urut-urutan aksi.

Berikut ini adalah program fungsional untuk realisasi dengan ide kalkulasi, yaitu ide yang ketiga, dengan hanya menuliskan ekspresi fungsional dasar, yang dapat menghasilkan nilai rata-rata 4 buah bilangan, tanpa bilangan terbesar dan tanpa bilangan terkecil yang diinginkan. 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)
DEFINISI DAN SPESIFIKASI	
<p>MO : 4 <u>integer</u> $\geq 0 \rightarrow$ <u>real</u></p> <p>{ <i>MO (u,v,w,x): menghitung rata-rata dari dua buah bilangan integer, yang bukan maksimum dan bukan minimum dari 4 buah integer:</i> $(u+v+w+x - \text{min4}(u,v,w,x) - \text{max4}(u,v,w,x)) / 2$ }</p> <p>max4 : 4 <u>integer</u> $> 0 \rightarrow$ <u>integer</u></p> <p>{ <i>max4 (i,j,k,l) menentukan maksimum dari 4 buah bilangan integer</i> }</p> <p>min4 : 4 <u>integer</u> $> 0 \rightarrow$ <u>integer</u></p> <p>{ <i>min4 (i,j,k,l) menentukan minimum dari 4 buah bilangan integer</i> }</p> <p>max2 : 2 <u>integer</u> $> 0 \rightarrow$ <u>integer</u></p> <p>{ <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>min2 : 2 <u>integer</u> $> 0 \rightarrow$ <u>integer</u></p> <p>{ <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>	
REALISASI	
<pre> max2 (a,b) : (a + b + abs(a - b)) / 2 min2 (a,b) : (a + b - abs(a - b)) / 2 max4 (i,j,k,l) : max2(max2(i,j),max2(k,l)) min4 (i,j,k,l) : min2(min2(i,j),min2(k,l)) MO (u,v,w,x) : (u+v+w+x - min4(u,v,w,x) - max4(u,v,w,x)) / 2 </pre>	
APLIKASI	
$\Rightarrow MO(8, 12, 10, 20)$	

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

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)
<u>DEFINISI DAN SPESIFIKASI</u>	
IsAnA? : <u>character</u> → <u>boolean</u> <i>{ IsAnA? (c) benar jika c adalah karakter (huruf) ‘A’ }</i>	
<u>REALISASI</u>	
IsAnA? (c) : $c = 'A'$	
<u>APLIKASI</u>	
$\Rightarrow \text{IsAnA?}('A')$ $\Rightarrow \text{IsAnA?}('B')$ $\Rightarrow \text{IsAnA?}('X')$	

Contoh-7 Ekspresi boolean : APAKAH ORIGIN

Persoalan :

Buatlah definisi, spesifikasi dan realisasi dan contoh aplikasi dari sebuah predikat yang menerima dua buah bilangan integer 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)
DEFINISI DAN SPESIFIKASI	
IsOrigin? : <u>integer</u> , <u>integer</u> → <u>boolean</u>	
<i>{ IsOrigin?(x,y) benar jika (x,y) adalah dua nilai yang mewakili titik origin (0,0) }</i>	
REALISASI	
IsOrigin?(x,y) : <u>x</u> = 0 <u>and</u> <u>y</u> = 0	
APLIKASI	
$\Rightarrow \text{IsOrigin?}(1,0)$	
$\Rightarrow \text{IsOrigin?}(1,1)$	
$\Rightarrow \text{IsOrigin?}(0,0)$	

Contoh-8 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)
DEFINISI DAN SPESIFIKASI	
IsValid? : <u>integer</u> → <u>boolean</u>	
<i>{ IsValid?(x) benar jika (x) bernilai lebih kecil 5 atau lebih besar dari 500 }</i>	
REALISASI	
IsValid?(x) : <u>x</u> < 5 <u>or</u> <u>x</u> > 500	
APLIKASI	
$\Rightarrow \text{IsValid?}(5)$	
$\Rightarrow \text{IsValid?}(0)$	
$\Rightarrow \text{IsValid?}(500)$	
$\Rightarrow \text{IsValid?}(6000)$	

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

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)

```
--> √ dif2(6,3) + dif2(5,1)      { pilih dif2(6,3)
                                         untuk dievaluasi dulu }

--> √ FX2(6-3) + dif2(5,1)      { expansi dif2(6,3) }

--> √ FX2(3) + dif2(5,1)        { reduksi, hasil evaluasi - }

--> √ 3 * 3 + dif2(5,1)        { ekspansi quad(3) }

--> √ 9 + dif2(5,1)            { reduksi * }

--> √ 9 + FX2(5-1)            { ekspansi dif2(5,1) }

--> √ 9 - FX2(4)              { reduksi - }

--> √ 9 + 4 * 4                { ekspansi quad(4) }

--> √ 9 + 16                  { reduksi * }

--> √ 25                      { reduksi + }

--> 5                          { reduksi √ }
```

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

Perhatikan urut-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 dan 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 independen 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-i>. Nama adalah lokal terhadap <NAMA-FUNGSI>. Nilai-nilai dan ekspresi <Ekspresi-i> hanya ada artinya di dalam <NAMA-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 independen (tak saling bergantung) satu sama lain.

Objektif analisa kasus adalah mendefinisikan partisi dari domain fungsi-fungsi yang akan merupakan solusi, dengan cara melakukan emunerasi dari semua kasus. Sebuah kasus adalah restriksi batasan dari problema dalam sebuah subdomain. Pada bahasa pemrograman, kasus seringkali disebut sebagai KONDISI, yang merupakan ekspresi bernilai boolean.

Menentukan kasus

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

- ada yang tidak tertulis,
- tidak *disjoint*.

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

Notasi Ekspresi Kondisional

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

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

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

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

artinya ekivalen dengan:

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

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

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

yang ekivalen dengan penulisan sebagai berikut :

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

Evaluasi Ekspresi Kondisional

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

```
(Kondisi-1 or Kondisi-2 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*, semua kondisi harus *mutually exclusive*.

Operator Boolean AND THEN dan OR ELSE

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

Operator AND then

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

Ekspresi boolean	Ekivalen dengan
A <u>AND then</u> B	<u>if</u> A <u>then</u> B <u>else</u> <u>false</u>

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</u> <u>true</u> <u>else</u> B

Catatan:

- Banyak bahasa hanya menyediakan if-then-else. Oleh karena itu, 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

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

dapat ditulis dalam bentuk lain yang ekivalen sebagai berikut:

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

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

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

dapat dituliskan:

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

Contoh-1 Ekspresi kondisional : MAKSIMUM 2 NILAI

Persoalan :

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

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

Contoh-2 Ekspresi kondisional : MAKSIMUM 3 NILAI

Persoalan :

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

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

Versi 1 : Identifikasi domain, berangkat dari hasil :

MAKSIMUM 3 NILAI (versi 1)	max3(a,b,c)
DEFINISI DAN SPESIFIKASI	
max3 : 3 integer → integer <i>{ max3(a,b,c) menentukan maksimum dari 3 bilangan integer yang berlainan nilainya, a ≠ b dan b ≠ c dan a ≠ c }</i>	

REALISASI

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

Versi 2 : Berdasarkan analisis letak ketiga bilangan pada sumbu bilangan, berangkat dari hasil. Karena data adalah 3 bilangan positif, dibagi sesuai posisi bilangan pada sumbu bilangan. Ada enam kemungkinan letak ke tiga nilai tersebut pada sumbu bilangan.

MAKSIMUM 3 NILAI (versi 2)

max3(a,b,c)

DEFINISI DAN SPESIFIKASI

max3 : 3 integer → integer

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

REALISASI

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

Versi 3 : Reduksi dari domain fungsi-fungsi.

- Ambil dua dari tiga nilai yang akan dicari maksimumnya, bandingkan.
- Manfaatkan hasil perbandingan untuk menentukan maksimum dengan cara membandingkan terhadap bilangan ketiga.

MAKSIMUM 3 NILAI (versi 3)

max3(a,b,c)

DEFINISI DAN SPESIFIKASI

max3 : 3 integer → integer

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

MAKSIMUM 3 NILAI (versi 4)	max3(a,b,c)
DEFINISI DAN SPESIFIKASI	
max3 : 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>	
REALISASI	
<pre>max3(a, b, c) let m = depend on (a, b): a > b : a a < b : b in depend on (m, c): m > c : m m < c : c</pre>	

Versi 5: Aplikasi suatu fungsi yang pernah dibuat.

Dengan sudah tersedianya MAX2, maka MAX3 dapat dituliskan dengan aplikasi dari MAX2, salah satu cara apalikasi adalah sebagai berikut :

MAKSIMUM 3 NILAI (versi 5)	max3 (a,b,c)
DEFINISI DAN SPESIFIKASI	
max3 : 3 <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>	
REALISASI	
<pre>max3(a, b, c) : max2(max2(a, b), c)</pre>	

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

MAKSIMUM 3 NILAI (versi 6)	max3 (a,b,c)
DEFINISI DAN SPESIFIKASI	
max3 : 3 <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>	
REALISASI	
<pre>max3(a, b, c) : max2(c, max2(a, b))</pre>	

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

Contoh-3 Ekspresi kondisional : PENANGGALAN

Persoalan :

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

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

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

Contoh : <1,1,82> → 1

 <31,12,72> → 366

 <3,4,93> → 93

Versi 1 : Tanpa memperhitungkan kabisat

PENANGGALAN	HariKe1900(d,m,y)
DEFINISI DAN SPESIFIKASI	
Harike1900 : integer [1..31], integer [1..12], integer [0..99] → integer [1..366] <i>{ Harike1900(d,m,y) dari suatu tanggal <d,m,y> adalah hari 'absolut' dihitung mulai 1 Januari 1900+y. 1 Januari tahun 1900+y adalah hari ke 1. }</i>	
dpm : integer [1..12] → integer [1..366] <i>{ dpm(B) adalah jumlah hari pada tahun ybs pada tanggal 1 bulan B. terhitung mulai 1 Januari: kumulatif jumlah hari dari tanggal 1 Januari s.d. tanggal 1 bulan B, tanpa memperhitungkan tahun kabisat. }</i>	
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 : <u>integer</u> [1..31], <u>integer</u> [1..12], <u>integer</u> [0..99] → <u>integer</u> [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 : <u>integer</u> [1..12] → <u>integer</u> [1..366]	
{ dpm(B) adalah jumlah hari pada tahun ybs pada tanggal 1 bulan B terhitung mulai 1 Januari: kumulatif jumlah hari dari tanggal 1 Januari s.d. tanggal 1 bulan B, tanpa memperhitungkan tahun kabisat }	
IsKabisat? : <u>integer</u> [0..99] → <u>boolean</u>	
{ 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) : ((1900+a <u>mod</u> 4 = 0) <u>and</u> (1900+a <u>mod</u> 100 ≠ 0)) <u>or</u> (1900+a <u>mod</u> 400 = 0)	
Harike1900 (d, m, y) :	
dpm(m) + d - 1 + (<u>if</u> m > 2 <u>and</u> IsKabisat?(y) <u>then</u> 1 <u>else</u> 0)	

Atau dapat dituliskan :

REALISASI
Harike1900 (d, m, y) : dpm (m) + d - 1 + (<u>if</u> m > 2 <u>and</u> then IsKabisat?(y) <u>then</u> 1 <u>else</u> 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,m,y) 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 integer $[0..24]$, m bilangan integer $[0..59]$, dan s bilangan integer $[0..59]$ yang artinya adalah jam, menit dan detik pada suatu tanggal tertentu. Hitunglah detik dari jam tersebut terhitung mulai jam 00:00:00 tanggal yang bersangkutan.
4. 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.
5. 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.
6. 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.
7. 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

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**, atau **record**. Untuk selanjutnya, dalam diktat ini disebut sebagai type bentukan.

Pokok bahasan dari bab ini adalah :

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

Pendefinisian Type Bentukan

Definisi type

Type adalah himpunan nilai dan sekumpulan operator yang 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 bertipe bentukan ditentukan oleh domain nilai komponennya. Karena merupakan *product*, nilai suatu type bentukan dituliskan dalam tuple, sesuai dengan komponen pembentuknya. Contoh : Untuk menyatakan **nilai** suatu Point yang didefinisikan oleh **tuple** $\langle x : \text{integer}, y : \text{integer} \rangle$, dipakai notasi : $\langle 0,0 \rangle$ sebagai Titik Origin, $\langle 1,2 \rangle$ untuk Titik dengan $x=1$ dan $y=2$.

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 $\langle \text{absis}, \text{ordinat} \rangle$ bertipe $\langle \text{integer}, \text{integer} \rangle$
2. Type Bilangan Kompleks, yang terdiri dari bagian riil dan bagian imaginer yang juga bertipe bilangan riil, sehingga memiliki tipe $\langle \text{real}, \text{real} \rangle$
3. Type Pecahan, yang terdiri dari $\langle \text{pembilang}, \text{penyebut} \rangle$ yang bertipe $\langle \text{integer}, \text{integer} \rangle$
4. Type JAM, yang terdiri dari $\langle \text{jam}, \text{menit}, \text{detik} \rangle$ dengan type $\langle \text{integer}, \text{integer}, \text{integer} \rangle$
5. Type DATE yang terdiri dari $\langle \text{tangal}, \text{bulan}, \text{tahun} \rangle$ bertipe $\langle \text{integer}, \text{integer}, \text{integer} \rangle$

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

1. Berdasarkan type Point $\langle \text{absis}, \text{ordinat} \rangle$, dapat dibentuk type berikut:
 - a) Garis, yang terdiri dari $\langle \text{titik-awal}, \text{titik-akhir} \rangle$.
 - b) Segiempat yang terdiri dari $\langle \text{titik-TopLeft}, \text{titik-BottomRight} \rangle$.

2. Berdasarkan JAM dan DATE, dapat dibentuk type baru WAKTU yang terdiri dari <jam,tanggal>.

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

Pendefinisian type bentukan dalam konteks fungsional adalah mendefinisikan:

- **Nama** type dan komposisi/strukturnya, hanya akan menjadi definisi
- **Selektor**, untuk mengakses komponen type komposisi sehingga menjadi elemen dasar yang dapat dioperasikan. Selektor dituliskan 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
- **Konstruktor** untuk “membentuk” type bentukan, 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
- **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 dalam 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 type bentukan.

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, dan 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 bentukan

Kasus 1 : Type POINT

Didefinisikan suatu type bernama Point, yang mewakili suatu titik dalam koordinat kartesian, terdiri dari absis dan ordinat.

Berikut ini adalah teks dalam notasi fungsional untuk type Point tersebut, dengan selektor yang hanya dituliskan dalam bentuk fungsi.

TYPE POINT

DEFINISI TYPE

type point : x: real , y: real >

{ *x,y* adalah sebuah point, dengan *x* adalah absis, *y* adalah ordinat }

DEFINISI DAN SPESIFIKASI SELEKTOR

Absis : point → real

{ *Absis(P)* memberikan Absis Point *P* }

Ordinat : point → real

{ *Ordinat(P)* memberikan ordinat Point *P* }

DEFINISI DAN SPESIFIKASI KONSTRUKTOR

MakePoint : 2 real → point

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

DEFINISI DAN SPESIFIKASI PREDIKAT

IsOrigin? : point → boolean

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

DEFINISI OPERATOR/FUNGSI LAIN TERHADAP POINT

Jarak : 2 point → real

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

Jarak0 : point → real

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

Kuadran : point → integer [1..4]

{ *Kuadran(P)* menghitung kuadran di mana titik tersebut terletak. Syarat: *P* bukan titik origin dan bukan terletak pada sumbu X atau pun sumbu Y }

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

REALISASI

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

Jarak (P1, P2) : SQRT (FX2(Absis(P1) - Absis(P2)) +
                     FX2(Ordinat(P1) - Ordinat (P2)))
Jarak0 (P) : SQRT (FX2(Absis(P)) + FX2(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 : $\langle n: \text{integer} \rangle \geq 0, d: \text{integer} > 0$

{ $\langle n: \text{integer} \geq 0, d: \text{integer} > 0 \rangle$. n adalah pembilang (numerator) dan d adalah penyebut (denominator). Penyebut sebuah pecahan tidak boleh nol. }

DEFINISI DAN SPESIFIKASI SELEKTOR DENGAN FUNGSI

Pemb : pecahan $\rightarrow \text{integer} \geq 0$

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

Peny : pecahan $\rightarrow \text{integer} > 0$

{ *Peny(p)* memberikan denominator/penyebut (d) dari pecahan tsb. }

DEFINISI DAN SPESIFIKASI KONSTRUKTOR

MakeP : $\text{integer} \geq 0, \text{integer} > 0 \rightarrow \text{pecahan}$

{ *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 $\rightarrow \text{pecahan}$

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

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., dengan syarat Pemb(P2) > 0
(pecahan pembagi bukan 0):
Membagi dua pecahan : $(n1/d1)/(n2/d2) = (n1*d2)/(d1*n2)$ }

RealP : pecahan → real

{ RealP(P) menuliskan pecahan P 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 sama nilainya: $n1/d1 = n2/d2$ jika dan hanya jika $n1*d2 = n2*d1$ }

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 $n1*d2 < n2*d1$ }

IsGtP?: 2 pecahan → boolean

{ IsGtP?(P1,P2) true jika $P1 > P2$:
Membandingkan dua buah pecahan,, apakah P1 lebih besar nilainya dari P2:
 $n1/d1 > n2/d2$ jika dan hanya jika $n1*d2 > n2*d1$ }

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 hari, 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) membentuk date 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 (<1,1,1980>) adalah <2,1,1980>
Nextday (<31,1,1980>) adalah <1,2,1980>
Nextday (<30,4,1980>) adalah <1,5,1980>
Nextday (<31,12,1980>) adalah <1,1,1981>
Nextday (<28,2,1980>) adalah <29,2,1980>
Nextday (<28,2,1981>) adalah <1,3,1982> }

Yesterday : date → date

{ *Yesterday(D)*: Menghitung date yang merupakan 1 hari sebelum date D yang diberikan:
 Yesterday (<1,1,1980>) adalah *<31,12,1979>*
 Yesterday (<31,1,1980>) adalah *<30,1,1980>*
 Yesterday (<1,5,1980>) adalah *<30,4,1980>*
 Yesterday (<31,12,1980>) adalah *<30,12,1980>*
 Yesterday (<29,2,1980>) adalah *<28,2,1980>*
 Yesterday (<1,3,1980>) adalah *<29,2,1980>* }

NextNDay : date, integer → date

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

HariKe : date → integer [0..366]

{ *HariKe(D)* : Menghitung jumlah hari terhadap 1 Januari pada tahun D.y, dengan memperhitungkan apakah D.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(<1,1,1990>,<1,1,1990>) adalah true
 EqD(<1,2,1990>,<1,1,1990>) adalah false }

IsBefore? : 2 date → boolean

{ *IsBefore?(d1,d2)* benar jika d1 adalah sebelum d2. Mengirimkan true jika D1 adalah "sebelum" D2: Contoh :
 Before(<1,2,1980>,<1,1,1980>) adalah false
 Before(<1,1,1980>,<2,1,1980>) adalah true }

IsAfter? : 2 date → boolean

{ *IsAfter?(d1,d2)* benar jika d1 adalah sesudah d2. Mengirimkan true jika D1 adalah "sesudah" D2. Contoh :
 After(<1,11,1980>,<1,2,1980>) adalah true
 After(<1,1,1980>,<2,1,1980>) adalah false
 After(<1,1,1980>,<1,1,1980>) adalah false }

IsKabisat? : Thn → boolean

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

REALISASI ISBEFORE DENGAN LET

```
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 ISBEFORE TANPA LET

```
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 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 : { Bulan dengan 30 hari }  
            if Day(D)<30 then MakeDate(Day(D)+1,Month(D),Year(D))  
            else MakeDate(Day(D),Month(D)+1,Year(D))  
        Month(D) = 12 { Desember }  
            if Day(D)<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 then { pindah ke bulan sebelumnya }
depend on Month(D)

Month(D) = 12 or Month(D) = 5 or Month(D) = 7
or Month(D) = 10 :
    MakeDate(30, Month(D)-1, Year(D))

Month(D) = 3 :
    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 or Month(D) = 2 or Month(D) = 8 :
    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) :

```
Year(D1) = Year(D2) and then HariKe(D1) = HariKe(D2)
```

IsBefore?(D1, D2) :

```
Year(D1) < Year(D2) or else
(Year(D1) = Year(D2) and HariKe(D1) < HariKe(D2))
```

IsAfter?(D1, D2) :

```
Year(D1) > Year(D2) or else
(Year(D1) = Year(D2) and HariKe(D1) > HariKe(D2))
```

IsKabisat?(a) :

```
((a mod 4 = 0) and (a mod 100 ≠ 0)) or (a mod 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.

Namun, ada kalanya 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 ke hari, jam, menit, detik

Diberikan sebuah besaran integer positif yang mewakili nilai detik. Tuliskanlah sebuah fungsi HHMMDD yang memberikan nilai hari, jam, menit, detik dari besaran detik tersebut.

Contoh: Diberikan 309639, menghasilkan $<3,14,0,39>$.

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

Realisasi fungsi ini membutuhkan sebuah fungsi yang mampu untuk menghitung hasil dan sekaligus sisa pembagian bulat dari dua buah bilangan integer yang dinamakan QR.

Range dari QR adalah sebuah pasangan nilai integer (type terkomposisi) namun tidak diberi nama.

EKIVALENSIDETIK	HHMMDD(x)
DEFINISI DAN SPESIFIKASI	
HHMMDD (x) : $\text{integer } [0..99999] \rightarrow <\text{integer} \geq 0, \text{integer}[0..23], 2 \text{ integer } [0..59]>$ { mis. $<a,b,c,d> = \text{HHMMDD } (x), x = 86400 * a + 3600 * b + 60 * c + d$ } QR : $<\text{integer}> 0, \text{integer}> 0 \rightarrow <\text{integer}> 0, \text{integer}> 0 >$ { let $<q,r> = \text{QR}(<N,D>)$: q adalah hasil bagi bulat r adalah sisa pembagian bulat dari N dibagi D $N = q * D + r$ and $0 < r < D$ }	
REALISASI	
QR (<N, D>) : $<N \text{ div } D, N \text{ mod } D>$ HHMMDD (x) : let $<H, \text{sisaH}> = \text{QR}(x, 86400)$ in let $<J, \text{sisaJ}> = \text{QR}(\text{sisaH}, 3600)$ in let $<M, D> = \text{QR}(\text{sisa J}, 60)$ in $<H, J, M, D>$	
REALISASI –VERSI TANPA LET	
QR (<N, D>) : $< N \text{ div } D, N \text{ mod } D >$ HHMMDD (x) : $< x \text{ div } 86400,$ $(x \text{ mod } 86400) \text{ div } 3600,$ $((x \text{ mod } 86400) \text{ mod } 3600) \text{ div } 60,$ $((x \text{ mod } 86400) \text{ mod } 3600) \text{ mod } 60 >$	

Perhatikanlah bahwa notasi $<a, b>$ artinya sebuah *tuple* yang membentuk sebuah “komposisi” dari nilai a dan b. Pendefinisian nilai bentukan semacam ini (dalam beberapa bahasa pemrograman disebut sebagai agregat) ternyata tidak disediakan primitifnya dalam semua bahasa pemrograman, berarti harus ditranslasi atau selalu dilakukan dengan

mendefinisikan nama type. Notasi fungsional membolehkan penulisan ini untuk kemudahan pembacaan teks program. Selanjutnya dalam implementasi harus diterjemahkan. Contoh terjemahan akan 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 sebagai type bentukan bernama.

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 : $\langle \text{bil} : \text{integer}, n : \text{integer}, d : \text{integer} \rangle$
dengan: bil : bilangan integer, mungkin bernilai 0 atau negatif
n : pembilang
d : penyebut
dan $n < d$ serta nilai pecahan adalah $(\text{bil} * d + n) / d$.
2. Realisasikan predikat IsEqD?, IsBefore?, dan IsAfter? tidak dengan melalui aplikasi HariKe, melainkan dengan melakukan analisa kasus terhadap $\langle d, m, y \rangle$.
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 point.
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 lingkaran: misalnya apakah sebuah titik terletak di dalam lingkaran, apakah lingkaran yang dibentuk berpusat pada $(0,0)$, apakah sebuah garis memotong lingkaran, menghitung garis tengah sebuah lingkaran, menghitung luas lingkaran.

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:

$$\begin{aligned}f(n) &= n^2 - n + 41 \\f(0) &= 41 & f(1) &= 41 & f(2) &= 43 \\f(3) &= 47 & f(4) &= 53\end{aligned}$$

$f(n)$ 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 di mana 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.

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 x sehingga $A=x*9$

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

Rekurens : untuk n sembarang, dianggap bahwa $10^n - 1$ habis dibagi 9 $\rightarrow 10^n - 1 = x*9$.

$$\begin{aligned}10^{n+1} - 1 &= 10 * (10^n - 1) + 9 \\&= 10 * (x * 9) + 9 = 90 * x + 9 \\&= 9 * (10x + 1)\end{aligned}$$

Terbukti bahwa $10^{n+1} - 1$ habis dibagi 9

Contoh-pembuktian-2

Buktikan bahwa semua bilangan bulat ≥ 2 dapat diuraikan dalam faktor primer, yaitu dapat diekspresikan sebagai proses 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, $k+1 = c * d$, $2 \leq c, d \leq k$.

Karena $2, 3, \dots, k$ mempunyai pembagi prima,

maka c dan d dapat ditulis dengan faktor bilangan prima, sehingga $k+1$ 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 P .

Maka:

P^a adalah himpunan seluruh himpunan bagian dari $E - P$ (hanya mengandung a)

P^1 adalah himpunan seluruh himpunan bagian dari P

Himpunan bagian $E - P$ dan P^1 adalah *disjoint*

Kardinalitas P^a adalah 2

Karena kardinalitas P adalah n , maka kardinalitas P^1 adalah 2^n

Himpunan bagian E dapat dibentuk dari hasil kartesian produk antara P^a dan P^1 , sehingga jumlah himpunan bagian dari E adalah $2 * 2^n = 2^{n+1}$.

Type Rekursif

Suatu type disebut type rekursif:

- Jika teks yang mendefinisikan type mengandung referensi terhadap diri sendiri, maka type tersebut adalah 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

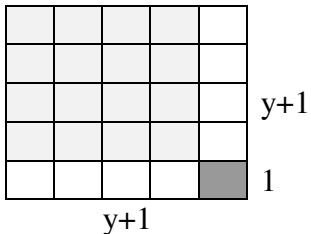
then $x + 2$ adalah bilangan integer ganjil

b. Luas bujur sangkar

Basis : luas bujur sangkar dengan sisi 1 = 1 satuan persegi

Rekurens :

Jika C adalah bujur sangkar dengan sisi y, \square
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 konstruktur

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

Konso(e,S) : e o S tambah sebuah elemen di 'kiri'

Kons•(S,e) : S•e tambah sebuah elemen di 'kanan'

d. Bilangan natural

Basis : 0 adalah bilangan natural

Rekurens :

Jika x adalah bilangan natural,

Bilangan natural yang lain dibentuk dengan menambahkan 1 pada x, $\text{succ}(x)$

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 langsung, 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 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-3>)  
  
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 di luar 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)
<u>DEFINISI DAN SPESIFIKASI</u>	
fac : <u>integer</u> $\geq 0 \rightarrow integer > 0$	
	{ $fac(n) = n!$ sesuai dengan definisi rekursif faktorial }
<u>REALISASI (VERSI-1)</u>	
{ Realisasi dengan definisi factorial sebagai berikut jika $fac(n)$ adalah $n!:$	
	$n = 0 : n! = 1$
	$n \geq 1 : n! = (n-1)! * n$
fac(n) :	<u>if</u> $n = 0$ <u>then</u> {Basis 0}
	1
	<u>else</u> {Rekurens : definisi faktorial}
	$fac(n-1) * n$
<u>REALISASI (VERSI-2)</u>	
{ Pada versi ini, domain dari fungsi berubah menjadi <u>integer</u> > 0	
Realisasi dengan definisi factorial sebagai berikut jika $fac(n)$ adalah $n!:$	
	$n = 1 : n! = 1$
	$n > 1 : n! = (n-1)! * n$
fac(n) :	<u>if</u> $n = 1$ <u>then</u> {Basis 1}
	1
	<u>else</u> {Rekurens : definisi faktorial}
	$n * fac(n-1)$
<u>REALISASI (VERSI-3): RUMUS BENAR, PROGRAM YANG SALAH !</u>	
{ Realisasi dengan definisi factorial sebagai berikut jika $fac(n)$ adalah $n!:$	
	$n = 1 : n! = 1$
	$n > 1 : n! = (n+1)! / n$
{ Realisasi berikut yang didasari definisi di atas tidak benar sebab evaluasi untuk $fac(n)$, $n > 1$ menimbulkan pemanggilan yang tidak pernah berhenti }	
fac(n) :	<u>if</u> $(n = 1)$ <u>then</u> {Basis 1}
	1
	<u>else</u> {Rekurens : definisi faktorial}
	$fac(n+1) / (n)$

Contoh-2 Ekspresi rekursif : FIBONACCI

Persoalan :

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

Fibonacci	Fib(n)
DEFINISI DAN SPESIFIKASI	
<p>Fib : <u>integer</u> $\geq 0 \rightarrow$ <u>integer</u> ≥ 0</p> <p>{ Definisi rekursif fungsi fibonacci : }</p> <p>{ Fib (n) = sesuai dengan definisi deret fibonacci :</p> <p style="padding-left: 20px;">$n = 0 : Fib(0) = 0$</p> <p style="padding-left: 20px;">$n = 1 : Fib(1) = 1$</p> <p style="padding-left: 20px;">$n > 1 : Fib(n) = Fib(n-1) + Fib(n-2)$ }</p>	
REALISASI	
<p>Fib(n) : <u>depend on</u> (n)</p> <p style="padding-left: 20px;">$n = 0 : 0$ {Basis 0}</p> <p style="padding-left: 20px;">$n = 1 : 1$ {Basis 1}</p> <p style="padding-left: 20px;">$n > 1 : Fib(n - 1) + Fib(n - 2)$ {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

DEFINISI KONSTRUKTOR

{ Konstruktor integer > 0 }

succ : integer $\geq 0 \rightarrow$ integer > 0

{ Basis: $n = 0 \rightarrow 0$

Rekurens : $n \rightarrow n + 1$

$succ(n)$ membentuk deret bilangan integer positif }

{ Konstruktor integer < 0 }

prec : integer \rightarrow integer

{ Basis: $n = \text{representasi maksimal atau minimal mesin.}$

Untuk n bilangan positif, basisnya seringkali diambil 0 }

{ Rekurens : $n \rightarrow n - 1$

$prec(n)$ membentuk deret bilangan integer negatif }

REALISASI

succ(n) : n + 1

prec(n) : n - 1

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

Contoh-3 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)
DEFINISI DAN SPESIFIKASI	
Exp : <u>integer</u> > 0, <u>integer</u> \geq 0 \rightarrow <u>integer</u> > 0	
{ <i>Exp</i> (x,y) menghasilkan x^y , $x \neq 0$ }	
{ <i>Exp</i> (x,y) = x^y	
= $x * x * x * x * \dots * x$	
= $x * x^{y-1}$ }	
{ Basis : $y = 0 \rightarrow 1$	
Rekurens : $y > 0 \rightarrow x * \text{Exp}(x, \text{prec}(y))$ }	
REALISASI	
Exp (x,y) :	
<u>if</u> y = 0 <u>then</u> {Basis 0}	
1	
<u>else</u> {Rekurens terhadap y}	
x * Exp(x, prec(y))	

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 jumlahnya, bisa banyak atau bisa sedikit

akan sangat sulit dalam mendefinisikan nama sebanyak yang dibutuhkan. Maka, dibutuhkan suatu fasilitas untuk mendefinisikan suatu nama kolektif, dan cara untuk mengakses setiap elemen.

Sekumpulan nilai disebut koleksi. Biasanya, suatu koleksi mungkin:

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

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

Operasi	Definisi domain dan range
Konstruktor untuk membuat koleksi kosong	→ Koleksi
Selektor, untuk melakukan akses terhadap elemen koleksi objek	Koleksi → elemen
Predikat untuk melakukan test: <ul style="list-style-type: none">▪ apakah koleksi kosong▪ apakah koleksi masih mampu menampung objek baru (belum penuh)	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 operasi 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 domain dan range
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.

LIST

Definisi

LIST adalah **sekumpulan** elemen list yang bertipe 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 bertipe 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 sehari-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:

- 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 Worksheets*, *list of cells*
- Dalam sistem operasi: *list of terminals*, *list of jobs*

Jika 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 diberikan realisasi 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 dua kelompok konstruktor dan selektor yang dituliskan dalam definisi list pada halaman berikutnya. 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 (lihat Diktat Bahasa LISP). 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.

TYPE LIST

DEFINISI DAN SPESIFIKASI TYPE

{ Konstruktor menambahkan elemen di awal }

type List : [], atau [e o List]

{ Konstruktor menambahkan elemen di akhir }

type List : [] atau [List • e]

{ Definisi List : sesuai dengan definisi rekursif di atas }

DEFINISI DAN SPESIFIKASI KONSTRUKTOR

Konso : elemen, List → List

{ Konso(e,L): menghasilkan sebuah list dari e dan L, dengan e sebagai elemen pertama
list: e o L → L'}

Kons• : List, elemen → List

{ Kons•(L,e): menghasilkan sebuah list dari L dan e, dengan e sebagai elemen terakhir
list: L • e → L'}

DEFINISI DAN SPESIFIKASI SELEKTOR

FirstElmt : List tidak kosong → elemen

{ FirstElmt(L) menghasilkan elemen pertama list L }

Tail : List tidak kosong → List

{ Tail(L) menghasilkan list L tanpa elemen pertama, mungkin kosong }

LastElmt : List tidak kosong → elemen

{ LastElmt(L) menghasilkan elemen terakhir list L }

Head : List tidak kosong → List

{ Head(L) menghasilkan list L tanpa elemen terakhir, mungkin kosong }

DEFINISI DAN SPESIFIKASI PREDIKAT DASAR

(UNTUK BASIS ANALISA REKURENS)

{ Basis 0 }

IsEmpty : List → boolean

{ IsEmpty(L) true jika list kosong }

{ Basis 1 }

IsOneElmt : List → boolean

{ IsOneElmt(L) true jika list L hanya mempunyai satu elemen }

DEFINISI DAN SPESIFIKASI PREDIKAT RELASIONAL

IsEqual : 2 List → boolean

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

BEBERAPA DEFINISI DAN SPESIFIKASI FUNGSI LAIN

NbElmt : List → integer

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

ElmtkeN : integer > 0 , List → elemen

{ *ElmtkeN(N, L)* mengirimkan elemen list yang ke-*N*, $0 < N \leq \text{NbElmt}(L)$ }

Copy : List → List

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

Inverse : List → List

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

Konkat : 2 List → List

{ *Konkat(L1, L2)* menghasilkan konkatenasi 2 list, dengan list *L2* "sesudah" list *L1* }

BEBERAPA DEFINISI DAN SPESIFIKASI PREDIKAT

IsMember : elemen, List → boolean

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

IsFirst : elemen, List tidak kosong → boolean

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

IsLast : elemen, List tidak kosong → boolean

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

IsNbElmtN : integer ≥ 0 , List → boolean

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

IsInverse : List, List → boolean

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

IsXElmtkeN : elemen, integer > 0, List → boolean

{ *IsXElmtkeN(X, N, L)* adalah benar jika *X* adalah elemen list yang ke-*N*, $0 < N \leq \text{NbElmt}(L)$ }

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(X, N, 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 lojik. Bagian ini akan merupakan salah satu bagian yang akan dirujuk dalam kuliah pemrograman lojik.

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 realisasinya 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:

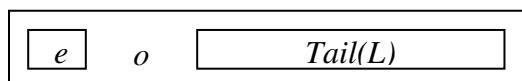
Basis : List kosong atau list 1 elemen

Rekurens : sebuah list terdiri dari elemen dan sisanya adalah list

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

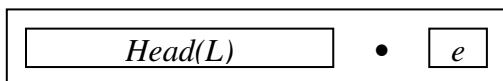
Dengan konstruktor Konso, ilustrasinya adalah:

Rekurens:



Dengan konstruktor Kons•, ilustrasinya adalah:

Rekurens:



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:

$$\text{NbElmt}([]) = 0 \quad \text{NbElmt}([a, b, c]) = 3$$

BANYAKNYA ELEMEN	NbElmt(L)
DEFINISI DAN SPESIFIKASI	
NbElmt : List \rightarrow integer	
$\{ \text{NbElmt}(L) \text{ menghasilkan banyaknya elemen list, nol jika list kosong} \}$	
$\{ \text{Basis 0: List kosong: tidak mengandung elemen, } \text{NbElmt}([]) = 0 \}$	
<i>Rekurens:</i>	
$\begin{array}{c} e \quad o \quad \boxed{\text{Tail}(L)} \\ \quad + \quad \text{NbElmt}(\text{Tail}(L)) \end{array}$	}
REALISASI: DENGAN KONSO	
NbElmt(L) :	
$\begin{array}{l} \text{if } \text{IsEmpty}(L) \text{ then } \{ \text{Basis 0} \\ \quad 0 \\ \text{else } \{ \text{Rekurens} \\ \quad 1 + \text{NbElmt}(\text{Tail}(L)) \end{array}$	

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:

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

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

KEANGGOTAAN	IsMember(x,L)
DEFINISI DAN SPESIFIKASI	
<p>IsMember : elemen, List \rightarrow boolean</p> <p><i>{ IsMember(x,L) true jika x adalah elemen dari list L }</i></p>	
REALISASI VERSI-1 : DENGAN KONSO	
<p><i>{ Basis 0 : List kosong: tidak mengandung apapun \rightarrow false</i></p> <p><i>Rekurens:</i> $\boxed{e} \text{ o } \boxed{\text{Tail}(L)}$ $? x$</p> <p><i>Kasus:</i> $e=x \rightarrow \text{true}$ $e \neq x \rightarrow x \text{ adalah anggota Tail}(L)? \}$</p> <pre> IsMember(x,L) : <u>if</u> IsEmpty(L) <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> <u>else</u> IsMember(x,Tail(L))</pre> <p style="text-align: right;">{ATAU}</p> <pre> IsMember(x,L) : <u>if</u> IsEmpty(L) <u>then</u> {Basis 0} <u>false</u> <u>else</u> {Rekurens: ekspresi boolean } FirstElmt(L)=x <u>or else</u> IsMember(x,Tail(L))</pre>	
REALISASI VERSI-2 : DENGAN KONS•	
<p><i>{ Basis 0: list kosong: tidak mengandung apapun \rightarrow false</i></p> <p><i>Rekurens:</i> $\boxed{\text{Head}(L)} \bullet \boxed{e}$ $? x$</p> <p><i>Kasus:</i> $e=x \rightarrow \text{true}$ $e \neq x \rightarrow x \text{ adalah anggota Head}(L)? \}$</p> <pre> IsMember(x,L) : <u>if</u> IsEmpty(L) <u>then</u> {Basis 0} <u>false</u> <u>else</u> {Rekurens: analisa kasus } <u>if</u> LastElmt(L)=x <u>then</u> <u>true</u> <u>else</u> IsMember(x,Head(L))</pre>	

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:

$$\text{Copy}([]) = [] \quad \text{Copy}([a, b, c]) = [a, b, c]$$

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

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:

$$\text{Inverse}([]) = [] \quad \text{Inverse}([a, b, c]) = [c, b, a]$$

MEMBALIK LIST	Inverse(L)
DEFINISI DAN SPESIFIKASI	
Inverse : List \rightarrow List	
<i>{ Inverse(L) menghasilkan salinan list L dengan urutan elemen terbalik }</i>	
<i>{ Basis 0: list kosong: hasilnya list kosong }</i>	
<i>Rekurens:</i>	
$\boxed{e} \ o \ \boxed{\text{Tail}(L)}$	
<i>Hasil pembalikan adalah Inverse(Tail(L)) • e }</i>	
REALISASI: DENGAN KONSO	
Inverse(L) :	
<i>if</i> IsEmpty(L) <i>then</i> {Basis 0}	
<i> []</i>	
<i>else</i> {Rekurens}	
<i> Kons•(Inverse(Tail(L)), FirstElmt(L))</i>	

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)
DEFINISI DAN SPESIFIKASI	
IsEqual : 2 List → <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> <i>IsEqual ([], []) = <u>true</u></i> <i>IsEqual ([], e o L) = <u>false</u></i> <i>IsEqual (e o K, []) = <u>false</u></i>	
<i>Rekurens</i> : terhadap kedua list <i>IsEqual (e1 o K1, e2 o K2) = (e1 = e2) dan (K1 = K2) }</i>	
REALISASI VERSI-1	
IsEqual (L1, L2) : {jika hanya salah satu yang kosong, maka false} depend on L1, L2 IsEmpty(L1) and IsEmpty(L2) : <u>true</u> {Basis} IsEmpty(L1) and <u>not</u> IsEmpty(L2) : <u>false</u> {Basis} <u>not</u> IsEmpty(L1) and IsEmpty(L2) : <u>false</u> {Basis} <u>not</u> IsEmpty(L1) and <u>not</u> IsEmpty(L1) : {Rekurens} (FirstElmt(L1) = FirstElmt(L2)) and then IsEqual(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 NbElmt(L1) ≠ NbElmt(L2) then <u>false</u> else CekEqual(L1, L2)	
CekEqual (L1, L2) : (IsEmpty(L1) and IsEmpty(L2)) or else 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 ≤ banyaknya elemen list. Contoh aplikasi dan hasilnya:

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

$\text{ElmtKeN}(1, [a,b,c]) = a$

Rekurens dilakukan terhadap N, dan juga terhadap list.

ELEMEN KE N	ElmtKeN(N,L)
DEFINISI DAN SPESIFIKASI	
<p>ElmtKeN : <u>integer</u> > 0, List tidak kosong → elemen</p> <p>{ <i>ElmtKeN(N,L)</i> menghasilkan elemen ke-N dari list <i>L</i>, $N > 0$, dan $N \leq$ banyaknya elemen list <i>L</i>. }</p> <p>{ Basis 1 : List dengan satu elemen, dan $N=1$: elemen pertama list</p>	
\boxed{e} <p>Rekurens:</p> $\boxed{e} \text{ o } \boxed{\text{Tail}(L)}$ $1 + \frac{\text{---}}{N-1}$ $\text{---} \text{ N} \text{ ---}$ <p><i>Kasus</i> : $N=1$ maka <i>e</i> $N>1$: bukan <i>e</i>, tetapi <i>ElmtKeN(N-1, Tail(L))</i> }</p>	
REALISASI: DENGAN KONSO	
<pre>ElmtKeN(N, L) : <u>if</u> N=1 <u>then</u> {Basis 1} FirstElmt(L) <u>else</u> {Rekurens} ElmtKeN(prec(N), Tail(L))</pre>	

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:

$\text{Konkat}([], []) = []$

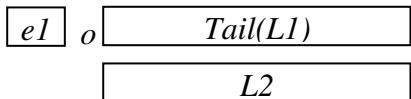
$\text{Konkat}([a], [b,c]) = [a,b,c]$

KONKATENASI	Konkat(L1,L2)
DEFINISI DAN SPESIFIKASI	
<p>Konkat : List → List</p> <p>{ <i>Konkat(L1,L2)</i> menghasilkan konkatenasi list <i>L1</i> dan <i>L2</i> }</p>	

REALISASI VERSI-1: REKURENS TERHADAP L1

{ Basis 0: $L1 = [] : L2$

Rekurens:



List Hasil: $e1 \circ \text{Hasil konkatenasi dari } Tail(L1) \text{ dengan } L2$

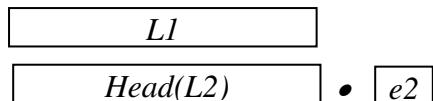
Konkat(L1, L2) :

```
if IsEmpty(L1) then {Basis 0}
L2
else {Rekurens}
  Konso(FirstElmt(L1), Konkat(Tail(L1), L2))
```

REALISASI VERSI-2: REKURENS TERHADAP L2

{ Basis 0: $L2 = [] : L1$

Rekurens:



List Hasil: Hasil konkatenasi dari $L1$ dengan $Head(L2) \bullet e2$

Konkat(L1, L2) :

```
if IsEmpty(L2) then {Basis 0}
L1
else {Rekurens}
  Kons•(Konkat(L1, Head(L2)), LastElmt(L2))
```

Contoh 8: Apakah banyaknya elemen sebuah list adalah N

Persoalan: Tuliskanlah definisi, spesifikasi, dan realisasi sebuah predikat yang memeriksa apakah banyaknya elemen sebuah list = N, dengan $N \geq 0$. Contoh aplikasi dan hasilnya:

$\text{IsNbElmtN}(0,[]) = \text{true}$ $\text{IsNbElmtN}(3,[a,b,c]) = \text{true}$

$\text{IsNbElmtN}(0,[a]) = \text{false}$

Rekurens dilakukan terhadap N.

BANYAKNYA ELEMEN

IsNbElmtN(L)

DEFINISI DAN SPESIFIKASI

IsNbElmtN : List, integer $\geq 0 \rightarrow \text{integer}$

{ $\text{IsNbElmtN}(L,N)$ true jika banyaknya elemen list sama dengan N }

REALISASI VERSI-1: dengan konso

{ Basis 0: List kosong: $N=0$?
 Rekurens:
$$\boxed{e} \quad o \quad \boxed{\text{Tail}(L)}$$

 # elemen: $1 \quad (N-1) \quad \}$

```
IsNbElmtN(L, N) :
    if N=0 or IsEmpty(L) then {Basis 0}
        N=0 and IsEmpty(L)
    else {Rekurens}
        IsNbElmtN(Tail(L), prec(N))
```

REALISASI VERSI-2: memanfaatkan fungsi NbElmt

```
IsNbElmtN(L, N) :
    NbElmt(L)=N
```

Contoh 9: Apakah X adalah elemen ke N sebuah list

Persoalan: Tuliskanlah definisi, spesifikasi, dan realisasi sebuah predikat yang memeriksa apakah X adalah elemen ke N dari sebuah list. N positif dan bernilai 1 s/d banyaknya elemen list, dan list tidak boleh kosong. Contoh aplikasi dan hasilnya:

IsXElmtKeN(b,1,[a,b,c]) = false IsXElmtKeN(c,3,[a,b,c]) = true

APAKAH X ELEMEN KE N	IsXElmtKeN(X,N,L)
DEFINISI DAN SPESIFIKASI	
IsXElmtKeN : elemen, <u>integer</u> > 0, List tidak kosong \rightarrow boolean	
{ IsXElmtKeN(X,N,L) true jika X adalah elemen ke-N list L, $N > 0$, dan $N \leq$ banyaknya elemen list }	
<u>REALISASI VERSI-1: dengan konso</u>	
{ Basis 1: List dengan satu elemen , dan $N=1$ dan $e=X$: true $\boxed{e} \quad e=X?$ Rekurens: $\boxed{e} \quad o \quad \boxed{\text{Tail}(L)}$ $\begin{array}{c} \hline \text{-----} \\ \text{N-1} \\ \hline \text{-----} \\ \text{N} \\ \hline \end{array}$ $\text{IsXElmtKeN}(X,N-1,\text{Tail}(L)) \quad \}$	IsXElmtKeN(X,N,L) : <u>if</u> N = 1 <u>then</u> {Basis 1} FirstElmt(L)=X <u>else</u> {Rekurens; Tail(L) pasti tidak kosong, sesuai domain N} IsXElmtKeN(X, prec(N), Tail(L))
<u>REALISASI VERSI-2: memanfaatkan fungsi ElmtKeN</u>	
IsXElmtKeN(X, N, L) : ElmtKeN(N, L)=X	

Contoh 10: Apakah inverse

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

APAKAH INVERSE	IsInverse(L1,L2)
DEFINISI DAN SPESIFIKASI	
IsInverse : 2 List → boolean	
<i>{ IsInverse(L1,L2) true jika L2 adalah list dengan urutan elemen terbalik dibandingkan L1, dengan perkataan lain adalah hasil inverse dari L1 }</i>	
REALISASI VERSI-1: DENGAN FUNGSI ANTARA	
IsInverse (L1, L2) : IsEqual(L1, Inverse(L2))	
REALISASI VERSI-2	
<i>{ Basis 0 : list kosong : true</i> <i>Rekurens: dua buah list sama, jika panjangnya sama dan</i>	
<i>L1 : e_1 o $Tail(L1)$</i>	
<i>L2 : $Head(L2)$ • x_2</i>	
<i>$e_1 = x_2$ dan $Tail(L1) = Inverse(Head(L2))$ }</i>	
IsInverse (L1, L2) : <i>if</i> IsEmpty(L1) <u>or</u> IsEmpty(L2) <i>then</i> {Basis 0} IsEmpty(L1) <u>and</u> IsEmpty(L2) <i>else</i> {Rekurens} (FirstElmt(L1)=LastElmt(L2)) <u>and</u> then IsInverse(Tail(L1), Head(L2))	

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 teks : $e \circ T \rightarrow T'$ }

Kons• : Teks, character → Teks

{ Kons•(T, e) menghasilkan sebuah list dari T dan e , dengan e sebagai elemen terakhir teks : $T \bullet e \rightarrow T'$ }

DEFINISI DAN SPESIFIKASI SELEKTOR

FirstChar: Teks tidak kosong → character

{ FirstChar(T) menghasilkan character pertama Teks T }

Tail : Teks tidak kosong → Teks

{ Tail(T) menghasilkan teks tanpa character pertama Teks T }

LastChar : Teks tidak kosong → character

{ LastChar(T) menghasilkan character terakhir Teks T }

Head : Teks tidak kosong → Teks

{ Head(T) menghasilkan teks tanpa character terakhir Teks T }

DEFINISI DAN SPESIFIKASI PREDIKAT DASAR

(UNTUK BASIS ANALISA REKURENS)

{ Basis 0 }

IsEmpty : Teks → boolean

{ IsEmpty(T) true jika list kosong }

{ Basis 1 }

IsOneElmt: Teks → boolean

{ IsOneElmt(T) true 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)
DEFINISI DAN SPESIFIKASI	
<p>nba : Teks \rightarrow <u>integer</u> ≥ 0</p> <p>{ <i>nba(T)</i> menghasilkan banyaknya kemunculan ‘a’ dalam teks <i>T</i> }</p>	
REALISASI VERSI-1: DENGAN KONS•	
<p>{ Basis 0: teks kosong: tidak mengandung ‘a’, <i>nba</i> ([]) = 0</p> <p>Rekurens:</p> <div style="display: flex; align-items: center; justify-content: center;"> <div style="border: 1px solid black; padding: 2px; margin-right: 10px;">Head(<i>T</i>)</div> • <div style="border: 1px solid black; padding: 2px; margin-right: 10px;"><i>e</i></div> <div style="margin-right: 20px;"><i>e=a?</i></div> </div> <p>ekspresikan domain berdasarkan elemen lain, selain Basis <i>nba (awt •LastChar)</i> bisa dievaluasi kalau <i>nba (awt)</i> diketahui :</p> <p style="text-align: center;"><i>nba(T)=nba(Head(T)) + 1 jika e adalah ‘a’</i> <i>nba(T) =nba(Head(T)) jika e bukan ‘a’ }</i></p>	
<pre>nba (T) : <u>if</u> IsEmpty(T) <u>then</u> {Basis 0} 0 <u>else</u> {Rekurens} nba(Head(T)) + <u>if</u> (LastChar(T)='a' <u>then</u> 1 <u>else</u> 0)</pre>	
REALISASI VERSI-2 : DENGAN KONSO	
<p>{ Basis 0 : teks kosong: tidak mengandung ‘a’, <i>nba</i> ([]) = 0</p> <p>Rekurens:</p> <div style="display: flex; align-items: center; justify-content: center;"> <div style="border: 1px solid black; padding: 2px; margin-right: 10px;"><i>e</i></div> o <div style="border: 1px solid black; padding: 2px; margin-right: 10px;">Tail(<i>T</i>)</div> <div style="margin-right: 20px;"><i>e=a?</i></div> </div> <p>ekspresikan domain berdasarkan elemen lain, selain Basis <i>nba(T)= 1 + nba(Tail(T)) jika e adalah ‘a’</i> <i>nba(T) = nba(Tail(T)) jika e bukan ‘a’ }</i></p>	
<pre>nba (T) : <u>if</u> IsEmpty(T) <u>then</u> {Basis 0} 0 <u>else</u> {Rekurens} <u>if</u> (FirstChar(T)='a' <u>then</u> 1 <u>else</u> 0) + nba(Tail(T))</pre>	
REALISASI VERSI-3 : BASIS BUKAN TEKS KOSONG	
<p>{ Basis 1:teks dengan satu karakter : <div style="border: 1px solid black; padding: 2px; margin-right: 10px;"><i>e</i></div> <i>e=a?</i></p> <p style="text-align: center;"><i>jika karakter adalah a, maka 1. Jika bukan ‘a’, maka 0</i></p> <p>Rekurens:</p> <div style="display: flex; align-items: center; justify-content: center;"> <div style="border: 1px solid black; padding: 2px; margin-right: 10px;">Head(<i>T</i>)</div> • <div style="border: 1px solid black; padding: 2px; margin-right: 10px;"><i>e1</i></div> • <div style="border: 1px solid black; padding: 2px; margin-right: 10px;"><i>e2</i></div> • <div style="margin-left: 20px;"><i>a</i></div> </div> <p>ekspresikan domain berdasarkan elemen lain, selain Basis <i>nba (awt . e1 . e2) = nba (awt . e1) + if(e2 = 'a') then 1 else 0</i> <i>selalu terhadap teks tidak kosong. }</i></p>	

```
nba (T) :
  if lastChar (T) = 'a' then 1 else 0) +
  (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

nbc(C,T)

DEFINISI DAN SPESIFIKASI

nbc : character, Teks → integer ≥ 0

```
{ nbc(C,T) menghitung banyaknya kemunculan karakter C pada teks T }
{ Basis 0 : nbc(C, [ ]) = 0
  teks kosong tidak mengandung karakter apapun }
{ Rekurens : nbc (C, T • e) :
  jika e = C maka, 1 + kemunculan karakter pada T
  jika e ≠ C, maka kemunculan karakter pada T }
```

REALISASI

```
nbc (C, T) :
  if IsEmpty(T) then {Basis 0}
  0
  else {Rekurens}
    if LastChar(T)=C then 1 + nbc(C, Head(T))
    else nbc(C, Head(T))
```

Contoh 3 Teks: Suatu karakter muncul pada teks T minimal N kali

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(N,C,T)

DEFINISI DAN SPESIFIKASI

Atleast : integer > 0 , character, teks → boolean

```
{ Atleast (N,C,T) true, jika karakter C minimal muncul N kali pada Teks T }
{ Basis 0 :
  (1) Atleast(0,C,[ ]) = true
    karakter C pada teks kosong akan muncul 0 kali
  (2) Atleast(0,C,C1oT) = true
    karakter apapun minimal muncul 0 kali pada teks selalu benar
  (3) Atleast(1+N,C,[ ]) = false, N bil. natural }
{ Rekurens :
  (4) Atleast(1+N,C,C1oT) = if (C=C1) then Atleast (N,C,T) else Atleast (1+N,C,T) }
```

REALISASI

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

Contoh 4 Teks: Palindrom

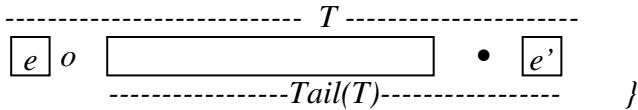
Persoalan: Tuliskanlah definisi, spesifikasi, dan realisasi dari sebuah predikat yang memeriksa apakah sebuah teks palindrom. Sebuah teks disebut palindrom jika dibaca dari awal sampai akhir identik dengan dibaca dari karakter akhir sampai awal. Contoh teks palindrom: “NABABAN”, “KASUR RUSAK”, “KASUR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

Memeriksa palindrom	IsPalindrome(T)
---------------------	-----------------

DEFINISI DAN SPESIFIKASI

IsPalindrome : Text → boolean

{ *IsPalindrome(T)* benar jika teks *T* adalah palindrom: jika dibaca dari kiri ke kanan, hasilnya sama dengan jika dibaca dari kanan ke kiri }
{ Basis 0 : teks kosong adalah palindrom
Basis 1 : teks dengan satu elemen adalah palindrom
Rekurens :



REALISASI DENGAN KONSO

```
IsPalindrome(T) :
  depend on (T)
    IsEmpty(T) : true {Basis-0}
    IsOneElmt(T) : true {Basis-1}
  else : {Rekurens}
    (FirstChar(T) = LastChar(T)) and then
    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 → List of integer

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

Kons• : List of integer, integer → List of integer

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

DEFINISI DAN SPESIFIKASI SELEKTOR

FirstElmt : List of integer tidak kosong → integer

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

Tail : List of integer tidak kosong → List of integer

{ *Tail*(*L*) menghasilkan list tanpa elemen pertama list *L* }

LastElmt : List of integer tidak kosong → integer

{ *LastElmt*(*L*) menghasilkan elemen terakhir list *L* }

Head : List of integer tidak kosong → 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 → boolean

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

{ Basis 1 }

IsOneElmt : List of integer → boolean

{ *IsOneElmt*(*L*) true jika list hanya mempunyai satu elemen }

DEFINISI DAN SPESIFIKASI PREDIKAT KEABSAHAN

IsListInt: List \rightarrow boolean

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

Berikut ini diberikan contoh persoalan yang spesifik terhadap pemrosesan elemen list yang bertipe 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 mengetes adalah predikat IsOneElmt, basis 1.

NILAI MAKSIMUM LIST INTEGER	MaxList(Li)
DEFINISI DAN SPESIFIKASI	
MaxList : List of <u>integer</u> tidak kosong \rightarrow <u>integer</u> { <i>MaxList(Li)</i> : menghasilkan elemen <i>Li</i> yang bernilai maksimum }	
REALISASI MaxList (Li) : <u>if</u> <i>IsOneElmt(Li)</i> <u>then</u> {Basis 1} LastElmt(Li) <u>else</u> {Rekurens} max2(<i>LastElmt(Li)</i> , <i>MaxList(Head(Li))</i>) { dengan <i>max2(a,b)</i> adalah fungsi yang mengirimkan nilai maksimum dari dua buah integer <i>a</i> dan <i>b</i> yang pernah dibahas sebelumnya }	

Contoh 2 List integer: Ukuran (Dimensi) List

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

UKURAN LIST	Dimensi(Li)
DEFINISI	
Dimensi : List of <u>integer</u> \rightarrow <u>integer</u> ≥ 0 { <i>Dimensi(Li)</i> menghasilkan banyaknya elemen list integer <i>Li</i> } { Basis 0 : dimensi list kosong = 0 Rekurens : dimensi (Li) $\boxed{e} \quad 0 \quad \boxed{\text{Tail}(Li)}$ $1 \quad + \quad \text{Dimensi}(\text{Tail}(Li)) \quad \}$	

REALISASI

```
Dimensi(Li) :  
    if IsEmpty(Li) {Basis} then 0  
    else {Rekurens} 1 + Dimensi(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

ListPlus(L1,L2)

DEFINISI

ListPlus : 2 list of integer $\geq 0 \rightarrow$ list of integer ≥ 0

```
{ ListPlus(Li1,Li2):  
  menjumlahkan setiap elemen list integer yang berdimensi sama, hasilnya berupa  
  list integer berdimensi tsb. }  
{ Basis 0 : Dimensi(Li1)=0 and Dimensi(Li2)=0 : []  
 Rekurens : Dimensi(Li1) ≠ 0 and Dimensi(Li2) ≠ 0 :  
   [e1] o [Tail(Li1)]  
   [e2] o [Tail(Li2)] +  
   _____  
   e1+e2 o ListPlus(Tail(Li1),Tail(Li2)) }
```

REALISASI

```
ListPlus(Li1, Li2) :  
    if Dimensi(Li1) = 0 then {Basis 0}  
    []  
    else {Rekurens}  
        Konso(FirstElmt(Li1)+FirstElmt(Li2),  
              ListPlus(Tail(Li1),Tail(Li2)))
```

Contoh 4 List 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	Insort(Li)			
DEFINISI DAN SPESIFIKASI				
Insort : list of <u>integer</u> → list of <u>integer</u> { <i>Insort(Li)</i> menghasilkan list integer yang terurut dengan metoda insertion sort } { Basis 0 : list kosong sudah terurut, hasilnya list kosong Rekurens : <i>Insort (Li)</i> <table border="1"><tr><td><i>e</i></td><td>o</td><td><i>Tail(Li)</i></td></tr></table> Hasil: Insert <i>e</i> ke <i>Tail(Li)</i> yang sudah terurut dg Insertion sort }		<i>e</i>	o	<i>Tail(Li)</i>
<i>e</i>	o	<i>Tail(Li)</i>		
Insert : <u>integer</u> , list of <u>integer</u> terurut membesar → list of <u>integer</u> terurut membesar { <i>Insert(x,Li)</i> melakukan insert <i>x</i> ke <i>Li</i> menghasilkan list terurut membesar } { Basis 0 : list kosong, insert elemen <i>x</i> ke list kosong : [x] Rekurens : <table border="1"><tr><td><i>e</i></td><td>o</td><td><i>Tail(Li)</i></td></tr></table> $x \leq e : x \text{ o } Li$ $x > e : e \text{ o } Insert(x, Tail(Li))$ Catatan : insert tidak pernah “merusak” keterurutan elemen list }		<i>e</i>	o	<i>Tail(Li)</i>
<i>e</i>	o	<i>Tail(Li)</i>		
REALISASI				
Insert (<i>x</i> , <i>Li</i>) : <u>if</u> IsEmpty (<i>Li</i>) <u>then</u> {Basis 0} Konso (<i>x</i> , <i>Li</i>) <u>else</u> {Rekurens} <u>if</u> (<i>x</i> ≤ FirstElmt (<i>Li</i>)) <u>then</u> Konso (<i>x</i> , <i>Li</i>) <u>else</u> Konso (FirstElmt (<i>Li</i>), Insert (<i>x</i> , Tail (<i>Li</i>))) Insort (<i>Li</i>) : <u>if</u> IsEmpty (<i>Li</i>) <u>then</u> {Basis 0} [] <u>else</u> {Rekurens} Insert (FirstElmt (<i>Li</i>), Insort (Tail (<i>Li</i>)))				

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 :

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

Nilai maks = m, #kemunculan m = n

Jika m adalah nilai maksimum dari 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)	maxNb(Li)
DEFINISI DAN SPESIFIKASI	
MaxNb : List of <u>integer</u> tidak kosong \rightarrow < <u>integer</u> , <u>integer</u> > <i>{ MaxNb(Li) menghasilkan <nilai maksimum, kemunculan nilai maksimum> dari suatu list integer Li: <m,n>; m adalah nilai maksimum di Li dan n adalah jumlah kemunculan m dalam list Li }</i>	
REALISASI VERSI-1	<pre>MaxNb(Li) : {menghasilkan nilai maksimum dan kemunculannya } if OneElmt(Li) then {Basis 1} <FirstElmt(Li),1> else {Rekurens} 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></pre>

Solusi versi-2 : dengan realisasi fungsi antara yang menghasilkan:

- Nilai maksimum
- Banyaknya kemunculan nilai maksimum

KEMUNCULAN MAKSIMUM (versi-2)	MaxNb(Li)
DEFINISI DAN SPESIFIKASI	
MaxNb : List of <u>integer</u> tidak kosong \rightarrow < <u>integer</u> , <u>integer</u> >	
<i>{ MaxNb(Li) menghasilkan <nilai maksimum, kemunculan nilai maksimum> dari suatu list integer Li: <m,n>; m adalah nilai maksimum di Li dan n adalah jumlah kemunculan m dalam list Li }</i>	
MaxList : List of <u>integer</u> tidak kosong \rightarrow <u>integer</u>	
<i>{ MaxList(Li) menghasilkan nilai maksimum dari elemen suatu list integer Li }</i>	
NbOcc : <u>integer</u> , List of <u>integer</u> tidak kosong \rightarrow <u>integer</u> > 0	
<i>{ NbOcc(X ,Li) yaitu banyaknya kemunculan nilai X pada Li }</i>	
REALISASI VERSI-2	
<pre>{ Realisasi MaxList telah dibahas di atas }</pre>	
NbOcc (X, Li) :	
<pre> 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))</pre>	
MaxNb (Li) :	
<pre><MaxList(Li) , NbOcc (MaxList(Li) , Li)></pre>	

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

TYPE SET (HIMPUNAN)

DEFINISI DAN SPESIFIKASI TYPE

{ *Set* adalah List dengan tambahan syarat bahwa tidak ada elemen yang sama }

{ Semua konstruktor, selektor dan fungsi pada List berlaku untuk Himpunan }

DEFINISI DAN SPESIFIKASI KONSTRUKTOR HIMPUNAN DARI LIST

{ Himpunan dibentuk dari list }

MakeSet : List → set

{ membuat sebuah set dari sebuah list }
{ yaitu membuang semua kemunculan yang lebih dari satu kali }
{ List kosong tetap menjadi list kosong }

DEFINISI DAN SPESIFIKASI PREDIKAT

IsSet : List → boolean

{ *IsSet(L)* true jika *L* adalah set }

IsSubSet : 2 set → boolean

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

DEFINISI DAN SPESIFIKASI OPERASI TERHADAP HIMPUNAN

MakeIntersect : 2 set → set

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

MakeUnion : 2 set → set

{ *Union (H1,H2)* membuat union *H1* dengan *H2* : yaitu set baru dengan semua anggota elemen *H1* dan anggota *H2* }

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 Set: Menghapus SEBUAH elemen sebagai anggota list

Fungsi ini dibutuhkan untuk membentuk himpunan

HAPUS 1 ELEMEN	Rember(x,L)
-----------------------	--------------------

DEFINISI DAN SPESIFIKASI

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. }

{ Basis : list kosong \rightarrow list kosong
Rekurens :

$$\boxed{e1} \quad o \quad \boxed{\text{Tail}(L)}$$

e = *x* : hasil adalah *Tail(L)*,
e \neq *x* : *e1* o Hasil *rember(e, Tail(L))* }

REALISASI

```
Rember(x,L) :
    if IsEmpty(L) then {Basis}
        L
    else {Rekurens : analisa kasus}
        if FirstElmt(L)=x then Tail(L)
        else Konso(FirstElmt(L), Rember(x, Tail(L)))
```

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

HAPUS SEMUA ELEMEN	Multirember(x,L)
DEFINISI DAN SPESIFIKASI	
MultiRember : elemen, List → List	
<p>{ <i>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.</i> }</p> <p>{ <i>Basis : list kosong → list kosong</i></p> <p><i>Rekurens :</i></p>	
$ \begin{array}{c} x \\ \boxed{e} \quad o \quad \boxed{\text{Tail}(L)} \end{array} $ <p><i>e = x : hapus semua x dari Tail(L) ,</i></p> <p><i>e ≠ x : e1 o hasil penghapusan semua x dari Tail(L) }</i></p>	
REALISASI	
MultiRember (x, L) : <code> 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))) </code>	

Contoh 3 Set: Mengetes apakah sebuah list adalah himpunan

Persoalan: Tuliskanlah definisi, spesifikasi, dan realisasi dari sebuah predikat yang akan mengetes apakah sebuah list adalah Himpunan

APAKAH SET	IsSet(L)
DEFINISI DAN SPESIFIKASI	
IsSet : List → boolean	
<p>{ <i>Set(L) true jika L adalah set</i> }</p> <p>{ <i>Basis : list kosong adalah set</i></p> <p><i>Rekurens :</i></p> $ \boxed{e} \quad o \quad \boxed{\text{Tail}(L)} $ <p><i>merupakan set jika Tail(L) tidak mengandung e }</i></p>	
REALISASI VERSI-1	
IsSet (L) : <code> if IsEmpty(L) then {Basis: list kosong adalah himpunan kosong} true else {Rekurens : analisa kasus} if IsMember(FirstElmt(L),Tail(L)) then false else IsSet(Tail(L)) </code>	

REALISASI VERSI-2

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

REALISASI VERSI-3

```
IsSet(L) :  
  IsEmpty(L) or else not IsMember(FirstElmt(L), Tail(L))  
  and then IsSet(Tail(L))
```

Contoh 4 Set: 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.

MEMBENTUK SET (versi-1)

MakeSet (L)

DEFINISI DAN SPESIFIKASI

MakeSet : List \rightarrow set

{ *MakeSet(L)* membuat sebuah set dari sebuah list, yaitu membuang semua kemunculan yang lebih dari satu kali. List kosong tetap menjadi list kosong. }
{ *Basis* : list kosong : \rightarrow List kosong
Rekurens :

$$e \quad o \quad \boxed{\text{Tail}(L)}$$

Untuk setiap *e* :

e adalah Member dari *Tail(L)* : *MakeSet(Tail(L))*
e bukan Member dari *Tail(L)* : *e o MakeSet(Tail(L))* }

REALISASI

```
MakeSet(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)))
```

APLIKASI

```
 $\Rightarrow \text{MakeSet}([\text{apel}, \text{sirsak}, \text{per}, \text{mangga}, \text{apel}, \text{jeruk}, \text{sirsak}])$   
{ \text{Himpunan hasil adalah : } [\text{per}, \text{mangga}, \text{apel}, \text{jeruk}, \text{sirsak}] }
```

MEMBENTUK SET (versi-2)	MakeSet (L)			
DEFINISI DAN SPESIFIKASI				
MakeSet : List \rightarrow set				
<pre>{ MakeSet(L) membuat sebuah set dari sebuah list, yaitu membuang semua kemunculan yang lebih dari satu kali. List kosong tetap menjadi list kosong. } { Basis : list kosong : \rightarrow () } Rekurens : <table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td style="padding: 2px 10px;">e</td><td style="padding: 2px 10px;">o</td><td style="padding: 2px 10px;">Tail(L)</td></tr> </table> e o MakeSet(Tail(L)) dengan Tail(L) yg tidak lagi mengandung e }</pre>		e	o	Tail(L)
e	o	Tail(L)		
REALISASI				
<pre>MakeSet (L) : if IsEmpty(L) then {Basis } L else {Rekurens } Konso(FirstElmt(L), MakeSet(MultiRember(FirstElmt(L), Tail(L))))</pre>				
APLIKASI				
<pre>\Rightarrow MakeSet([apel, sirsak, per, mangga, apel, jeruk, sirsak]) { Himpunan hasil : [apel, sirsak, per, mangga, jeruk] }</pre>				

Contoh 5 Set: Mengetes apakah sebuah set merupakan subset dari set yang lain

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

APAKAH SUBSET	IsSubSet(H1,H2)						
DEFINISI DAN SPESIFIKASI							
IsSubSet : 2 set \rightarrow boolean							
<pre>{ 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. } { Basis : list kosong \rightarrow true Rekurens : H1 <table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td style="padding: 2px 10px;">e</td><td style="padding: 2px 10px;">o</td><td style="padding: 2px 10px;">Tail(H1)</td></tr> </table> H2 <table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td style="padding: 2px 10px;">e</td><td style="padding: 2px 10px;">o</td><td style="padding: 2px 10px;">Tail(H2)</td></tr> </table> 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 }</pre>		e	o	Tail(H1)	e	o	Tail(H2)
e	o	Tail(H1)					
e	o	Tail(H2)					

REALISASI

```
IsSubSet(H1, H2) :  
  if IsEmpty(H1) then true { Basis }  
  else { Rekurens: 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 Set: Mengetes kesamaan dua buah set

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

KESAMAAN DUA SET	IsEQSet (H1,H2)
DEFINISI PREDIKAT	
IsEQSet : 2 set → boolean	<i>{ 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 }</i>
REALISASI	
IsEQSet (H1, H2) : IsSubSet (H1, H2) and IsSubSet (H2, H1)	

Contoh 7 Set: Mengetes apakah dua buah set berinterseksi

Persoalan: Tuliskanlah definisi, spesifikasi, dan realisasi dari sebuah predikat yang akan mengetes apakah dua buah himpunan saling beririsan.

APAKAH INTERSEKSI	IsIntersect (H1,H2)
DEFINISI DAN SPESIFIKASI	
IsIntersect : 2 set → boolean	<i>{ 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. } { Basis : Salah satu kosong : → false Rekurens :</i>
<i>H1 [e1] o Tail(H1)</i>	
H2	<i>[H2]</i>
<i>e1 adalah Member dari H2 : true e1 bukan Member dari H2 : IsIntersect(Tail(H1),H2) }</i>	

REALISASI

```
IsIntersect(H1, H2) :  
  depend on H1, H2  
  IsEmpty(H1) or IsEmpty(H2) : false { Basis }  
  not IsEmpty(H1) and not IsEmpty(H2) : IsMember(FirstElmt(H1), H2) or else IsIntersect(Tail(H1), H2)
```

Contoh 8 Set: 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

BUAT INTERSEKSI

MakeIntersect(H1,H2)

DEFINISI DAN SPESIFIKASI

MakeIntersect : 2 set \rightarrow set

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

{ Basis : Jika salah satu kosong, hasilnya set kosong

Rekurens :

<i>H1</i>	<u><i>e1</i></u>	<u><i>o</i></u>	<u><i>Tail(H1)</i></u>
<i>H2</i>	<u><i>H2</i></u>		

e1 adalah member dari *H2* : *e1* o *MakeIntersect(Tail(H1),H2)*

e1 bukan member dari *H2* : *MakeIntersect(Tail(H1),H2)* }

REALISASI

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

Contoh 9 Set: 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)
DEFINISI DAN SPESIFIKASI	
MakeUnion : 2 set → set	
<p>{ <i>MakeUnion (H1,H2)</i> membuat union <i>H1</i> dengan <i>H2</i> : yaitu set baru dengan semua anggota elemen <i>H1</i> dan anggota <i>H2</i> }</p> <p>{ <i>Basis</i> : Jika salah satu kosong, hasilnya adalah himpunan yang tidak kosong Kedua set kosong, hasilnya sebuah set yang kosong</p>	
<i>Rekurens</i> :	
	$ \begin{array}{c} H1 \quad e1 \quad o \quad \boxed{\text{Tail}(H1)} \\ \hline H2 \quad \boxed{H2} \end{array} $
<p><i>e1</i> adalah Member dari <i>H2</i> : buang <i>e1</i>, <i>MakeUnion(Tail(H1),H2)</i></p> <p><i>e1</i> bukan Member dari <i>H2</i> : <i>e1 o MakeUnion(Tail(H1),H2)</i></p>	
REALISASI	
MakeUnion (H1, H2) :	
depend on H1, H2	
IsEmpty(H1) and IsEmpty(H2) : [] { Basis }	
not IsEmpty(H1) and IsEmpty(H2) : H1 { Basis }	
IsEmpty(H1) and not IsEmpty(H2) : H2 { Basis }	
not IsEmpty(H1) and not IsEmpty(H2) : { Rekurens }	
if IsMember(FirstElmt(H1), H2)	
then MakeUnion(Tail(H1), H2)	
else Konso(FirstElmt(H1), Union(Tail(H1), H2))	

Catatan:

Perhatikan kembali realisasi fungsi `IsIntersect(H1,H2)`, `MakeIntersect(H1,H2)`, dan `MakeUnion(H1,H2)`. Ketiga fungsi tersebut didefinisikan memiliki 3 buah basis, yaitu:

IsEmpty(H1) and IsEmpty(H2)
not IsEmpty(H1) and IsEmpty(H2)
IsEmpty(H1) and not IsEmpty(H2)

Pada kenyataannya, rekursif hanya dilakukan terhadap H1, sedangkan H2 tidak pernah diubah isinya untuk setiap kali pemanggilan. Oleh sebab itu, penempatan H2 sebagai basis proses rekursif sebenarnya tidak terlalu tepat. Bagaimana alternatif realisasi yang lebih tepat?

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. **List dalam sebuah list of list pada dasarnya adalah list of list juga.** 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

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

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

list dengan elemen list kosong, S1, S2 dan sebuah atom g

S1 adalah list dengan elemen 3 buah atom a, b, c

S2 adalah list dengan elemen sebuah atom d dan S3

S3 adalah list dengan elemen 2 buah atom e dan f

Untuk list khusus ini, nama konstruktor dan selektor tidak dibedakan dengan list yang hanya mengandung elemen dasar, namun diperlukan predikat tambahan untuk mengetahui apakah sebuah elemen/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 \rightarrow boolean

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

IsOneElmt : List of list \rightarrow boolean

{ *IsOneElmt(S) benar jika S adalah list of list dengan 1 elemen* }

IsAtom : **Atom/**List \rightarrow boolean

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

IsList : **Atom/**List \rightarrow boolean

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

DEFINISI DAN SPESIFIKASI KONSTRUKTOR

KonsLo : **Atom/**List, List of list \rightarrow List of list

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

KonsL• : List of list, **Atom/**List \rightarrow List of list

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

DEFINISI DAN SPESIFIKASI SELEKTOR

FirstList : List of list tidak kosong \rightarrow **Atom/**List

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

TailList : List of list tidak kosong \rightarrow List of list

{ *TailList(S) menghasilkan "sisa" list of list S tanpa elemen pertama list of list S* }

LastList : List of list tidak kosong \rightarrow **Atom/**List

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

HeadList : List of list tidak kosong \rightarrow List of list

{ *HeadList(S) menghasilkan "sisa" list of list S tanpa elemen terakhir list of list S* }

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.

Contoh:

$\text{IsEqS}([], []) = \text{true}$

$\text{IsEqS}([a], []) = \text{false}$

$\text{IsEqS}([], [a]) = \text{false}$

$\text{IsEqS}([[a,b],c], [[a,b],c]) = \text{true}$

KESAMAAN	$\text{IsEqS}(S1, S2)$
DEFINISI PREDIKAT	
IsEqS: 2 List of list \rightarrow boolean	
$\{ \text{IsEqS}(S1, S2) \text{ true jika } S1 \text{ identik dengan } S2: \text{ semua elemennya sama } \}$ $\{ \text{Basis} : \text{kedua list kosong} : \rightarrow \text{true}$ $\quad \text{salah satu list kosong} : \rightarrow \text{false}$	
<i>Rekurens :</i> $S1 \quad [L1] \circ \boxed{\text{Tail}(S1)}$ $S2 \quad [L2] \circ \boxed{\text{Tail}(S2)}$ <p><i>L1 dan L2 adalah atom : $L1=L2$ and $\text{IsEqS}(\text{TailList}(S1), \text{TailList}(S2))$</i> <i>L1 dan L2 adalah list : $\text{IsEqS}(S1, S2)$ and $\text{IsEqS}(\text{TailList}(S1), \text{TailList}(S2))$</i> <i>else : false }</i></p>	
REALISASI	
IsEqS ($S1, S2$) : <u>depend on</u> $S1, S2$ <ul style="list-style-type: none"> $\text{IsEmpty}(S1) \text{ and } \text{IsEmpty}(S2) : \text{true} \quad \{ \text{basis} \}$ $\text{not } \text{IsEmpty}(S1) \text{ and } \text{IsEmpty}(S2) : \text{false} \quad \{ \text{basis} \}$ $\text{IsEmpty}(S1) \text{ and } \text{not } \text{IsEmpty}(S2) : \text{false} \quad \{ \text{basis} \}$ $\text{not } \text{IsEmpty}(S1) \text{ and } \text{not } \text{IsEmpty}(S2) : \quad \{ \text{rekurens} \}$ <ul style="list-style-type: none"> <u>depend on</u> $\text{FirstList}(S1), \text{FirstList}(S2)$ <ul style="list-style-type: none"> $\text{IsAtom}(\text{FirstList}(S1) \text{ and } \text{IsAtom}(\text{FirstList}(S2)) : \quad \text{FirstList}(S1) = \text{FirstList}(S2) \text{ and }$ $\quad \text{IsEqS}(\text{TailList}(S1), \text{TailList}(S2))$ $\text{IsList}(\text{FirstList}(S1) \text{ and } \text{IsList}(\text{FirstList}(S2)) : \quad \text{IsEqS}(\text{FirstList}(S1), \text{FirstList}(S2)) \text{ and }$ $\quad \text{IsEqS}(\text{TailList}(S1), \text{TailList}(S2))$ <u>else</u> : {atom dengan list pasti tidak sama} <u>false</u> 	

Contoh 2: Mengecek apakah sebuah atom merupakan elemen sebuah list of list

Persoalan : Tuliskanlah definisi, spesifikasi, dan realisasi dari sebuah predikat yang mengecek keanggotaan sebuah elemen terhadap list of list.

KEANGGOTAAN	IsMemberS(A,S)
DEFINISI PREDIKAT	
IsMemberS : elemen, List of list → boolean	
$\{ \text{IsMemberS}(A, S) \text{ true jika } A \text{ adalah anggota } S \}$	
{ Basis : list kosong : → false	
Rekurens :	
$L1$	\bigcirc
$L1$	$\boxed{\text{Tail}(S)}$
$L1$ adalah atom dan $A = L1$: true	
$L1$ bukan atom : $\text{IsMemberS}(A, \text{FirstList}(S)) \text{ or } \text{IsMemberS}(A, \text{TailList}(S))$ }	
REALISASI	
IsMemberS (A, S) :	
depend on S	
IsEmpty(S) : false {basis}	
not IsEmpty(S) :	
depend on FirstList(S)	
IsAtom(FirstList(S)) :	
A = FirstList(S) {basis}	
or else IsMemberS(A, TailList(S)) {rekurens}	
IsList(FirstList(S)) : {rekurens}	
IsMemberS(A, FirstList(S))	
or else IsMemberS(A, TailList(S))	

Contoh 3: Mengecek apakah sebuah List merupakan elemen sebuah list of list

Persoalan: Tuliskanlah definisi, spesifikasi, dan realisasi dari sebuah fungsi yang mencek keanggotaan sebuah list terhadap list of list.

KEANGGOTAAN	IsMemberLS(L,S)
DEFINISI PREDIKAT	
IsMemberLS : List, List of list \rightarrow boolean	
$\{ \text{IsMemberLS}(L,S) \text{ true jika } L \text{ adalah anggota } S \}$	
$\{ \text{Basis} : S \text{ list kosong} : \rightarrow \text{false}$	
Rekurens :	
$L1 \circ TailList(S)$	
$L1$ adalah list dan $L = L1 : \text{true}$	
$L1$ adalah list dan $L \neq L1 : \text{IsMemberLS}(L, \text{FirstList}(S)) \text{ or } \text{IsMemberLS}(L, \text{TailList}(S))$	
$L1$ bukan list : $\text{IsMemberLS}(L, \text{TailList}(S))$	
REALISASI	
IsMemberLS (L, S) :	
depend on S	
IsEmpty(S) : <u>false</u> { basis }	
not IsEmpty(S) :	
depend on (FirstList(S))	
IsList(FirstList(S)) :	
IsEqS(L, FirstList(S)) { basis }	
or else (IsMemberLS(L, FirstList(S))) { rekurens }	
or else IsMemberLS(L, TailList(S)))	
IsAtom(FirstList(S)) : { rekurens }	
IsMemberLS(L, TailList(S))	

Contoh 4: Menghapus sebuah elemen (atom) dari list of list

Persoalan: Tuliskanlah definisi, spesifikasi, dan realisasi dari sebuah fungsi yang menghapus semua kemunculan atom tertentu pada sebuah list of list

Contoh:

$\text{Rember}^*(a, []) = []$

$\text{Rember}^*(a, [[b,c], a, d, [b,c, a]]) = [[b,c], d, [b,c]]$

HAPUS ELEMEN	$\text{Rember}^*(a, S)$
<p>DEFINISI</p> <p>Definisi: Rember^*: elemen, List of list \rightarrow List of list</p> <p>{ $\text{Rember}^*(a, S)$ menghapus semua kemunculan a pada list of list S. List kosong tetap menjadi list kosong }</p> <p>{ Basis : list kosong : $\rightarrow []$</p> <p>Rekurens :</p> <p style="padding-left: 40px;">a</p> <p style="padding-left: 40px;">$S \quad L1 \text{ o } \boxed{\text{Tail}(S)}$</p> <p>$L1$ adalah atom : $L1 = a : \text{TailList}(S)$ tanpa a $L1 \neq a : L1 \text{ o } (\text{TailList}(S)$ tanpa a) $L1$ adalah list : $(L1$ tanpa $a)$ o $(\text{TailList}(S)$ tanpa a)</p>	
<p>REALISASI</p> <p>Definisi: $\text{Rember}^*(a, S)$:</p> <pre>if IsEmpty(S) then S {basis} else if IsList(FirstList(S)) then KonsLo(Rember*(a, FirstList(S)), Rember*(a, TailList(S))) else { elemen pertama S adalah atom } if FirstList(S) = a then Rember*(a, TailList(S)) else KonsLo(FirstList(S), Rember*(a, TailList(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 dengan atom integer yang tidak kosong dengan elemen list yang (jika ada) juga tidak kosong.

ELEMEN BERNILAI MAKSIMUM	Max(S)
DEFINISI	
MaxList : List of list tidak kosong \rightarrow <u>integer</u>	
$\{ \text{MaxList}(S) \text{ menghasilkan nilai elemen (atom) yang maksimum dari } S \}$	
$\{ \text{Basis} : \text{list dengan satu elemen } E1$	
$E1 \text{ adalah atom} : E1$	
$E1 \text{ adalah list} : \text{Max}(E1)$	
<i>Rekurens :</i>	
$S = \boxed{L1} \circ \boxed{\text{TailList}(S)}$	
$L1 \text{ adalah atom} : \text{Max2}(L1, \text{Max}(\text{TailList}(S)))$	
$L1 \text{ adalah list} : \text{Max2}(\text{Max}(L1), \text{Max}(\text{TailList}(S)))$	}
<i>{ Fungsi antara }</i>	
Max2 : 2 <u>integer</u> \rightarrow <u>integer</u>	
$\{ \text{Max2}(a,b) \text{ menghasilkan nilai maksimum } a \text{ dan } b \}$	
REALISASI	
Max2 (a, b) :	
<u>if</u> $a \geq b$ <u>then</u>	
a	
<u>else</u>	
b	
MaxList (S) :	
<u>if</u> $\text{IsOneElmt}(S)$ <u>then</u> {Basis 1}	
<u>if</u> $\text{IsAtom}(\text{FirstList}(S))$ <u>then</u>	
FirstList(S)	
<u>else</u> {List, pasti tidak kosong sesuai spesifikasi }	
MaxList(FirstList(S))	
<u>else</u> {Rekurens}	
<u>if</u> $\text{IsAtom}(\text{FirstList}(S))$ <u>then</u> {First elemen adalah atom }	
Max2(FirstList(S), MaxList(TailList(S)))	
<u>else</u> { First element adalah List }	
Max2(MaxList(FirstList(S)), MaxList(TailList(S)))	

Resume Analisa Rekurens

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

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

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

Rekurens :

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

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

$\text{IsList}(\text{FirstList}(S)) : g(\text{ekspresi terhadap list of list}, f(\text{TailList}(S)))$

POHON

Pendahuluan

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

- mengorganisasi informasi berdasarkan suatu struktur “lojik”
- 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 diindentasi mencerminkan struktur pohon dari buku tersebut.

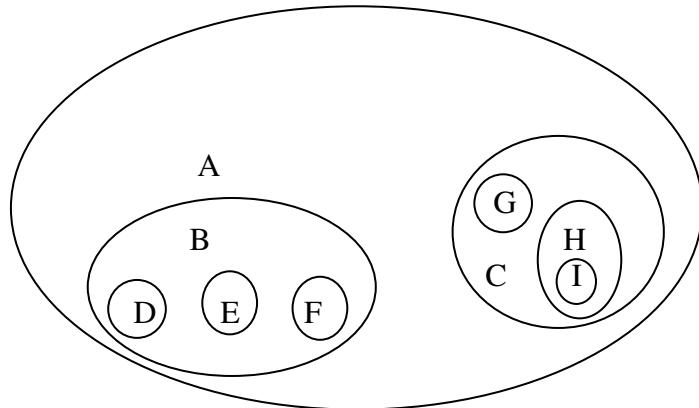
Catatan:

- Sufiks (akhiran) *n-air* menunjukkan bahwa jumlah sub pohon bervariasi dari 1 hingga *n* buah.
- Semua elemen dari pohon merupakan akar dari suatu 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.

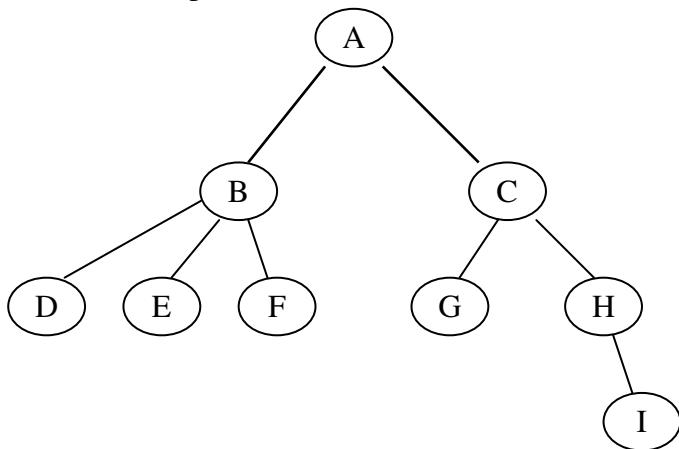
Cara Penulisan Pohon:

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

a) Himpunan yang saling melingkupi



b) Graph



c) Indentasi

```

A
B
D
E
F
C
G
H
I
  
```

d) Bentuk linier :

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

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

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

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

ORANG TUA/ORTU (*parent*): Akar dari sebuah pohon/sub pohon.

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

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

DAUN (*leaf*): adalah simpul terminal dari pohon (simpul tanpa anak). Semua simpul selain daun adalah simpul BUKAN-TERMINAL (*internal node*).

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

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

TINGKAT (*Level*): suatu simpul 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:

$$N = b + u + d$$

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

sehingga

$$b = d - 1$$

Representasi pohon n-airy (Pohon N-er) : 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-aire.

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

TYPE POHON-N-AIRE (tidak mungkin kosong)

DEFINISI DAN SPESIFIKASI TYPE

type elemen : { tergantung type node }

type PohonN-er : < A : elemen, PN : List of PohonN-er > { notasiPrefix }, atau

type PohonN-er : < PN : List of PohonN-er, A : elemen > { notasi postfix }

{ *Pohon N-er terdiri dari Akar yang berupa elemen dan list dari pohon N-er yang menjadi anaknya. List anak mungkin kosong, tapi pohon N-er tidak pernah kosong, karena minimal mempunyai sebuah elemen sebagai akar pohon* }

DEFINISI DAN SPESIFIKASI SELEKTOR

Akar : PohonN-er tidak kosong → elemen

{ *Akar(P) adalah Akar dari P. Jika P adalah (A,PN) maka Akar(P) = A.* }

Anak : PohonN-er tidak kosong → List of PohonN-er

{ *Anak(P) adalah list of pohon N-er yang merupakan anak-anak (sub pohon) dari P. Jika P adalah (A, PN) maka Anak (P) = PN* }

DEFINISI DAN SPESIFIKASI KONSTRUKTOR

{ *Perhatikanlah bahwa konstruktor pohon N-er dengan basis pohon tidak kosong dituliskan sebagai*

a. Prefix : (A,PN)

b. Posfix : (PN,A) }

DEFINISI DAN SPESIFIKASI PREDIKAT

IsOneElmt : PohonN-er → boolean

{ *IsOneElmt(PN) true jika PN hanya terdiri dari Akar* }

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

Pohon Biner

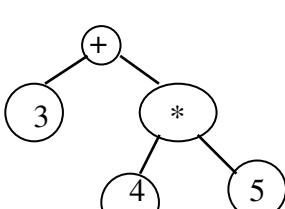
Definisi:

Sebuah pohon biner adalah himpunan terbatas yang

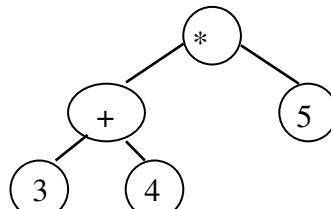
- mungkin kosong, atau
 - terdiri dari sebuah simpul yang disebut akar, dan
- dua buah himpunan lain yang *disjoint* yang merupakan **pohon biner**, yang disebut sebagai sub pohon kiri dan sub pohon kanan dari pohon biner tersebut

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

Contoh pohon ekspresi aritmatika

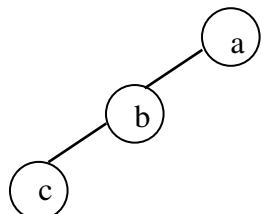


$3 + (4 * 5)$

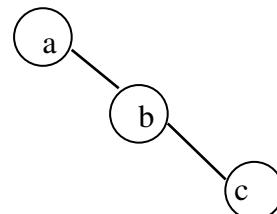


$(3+4) * 5$

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



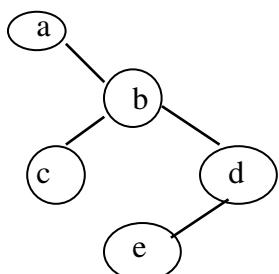
Pohon biner condong kiri



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 : <L : PohonBiner, A : elemen, R : PohonBiner> {notasi Infix }, atau

type PohonBiner : <A : elemen, L : PohonBiner, R : PohonBiner> {notasi prefix }, atau

type PohonBiner : <L : PohonBiner, R : PohonBiner, A : elemen> {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 → elemen

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

Left : PohonBiner tidak kosong → PohonBiner

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

Right : PohonBiner tidak kosong → 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. **Postfix** : //L R A\\ }

DEFINISI DAN SPESIFIKASI PREDIKAT

IsTreeEmpty : PohonBiner → boolean

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

DEFINISI DAN SPESIFIKASI PREDIKAT LAIN

NbElmt : PohonBiner \rightarrow integer ≥ 0

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

Basis : $\text{NbElmt}(\text{//}\backslash\backslash) = 0$

Rekurens : $\text{NbElmt}(\text{//}L,A,R\backslash\backslash) = \text{NbElmt}(L) + 1 + \text{NbElmt}(R)$ }

NbDaun : PohonBiner \rightarrow integer ≥ 0

{ *Definisi* : *Pohon kosong memiliki 0 daun* }

{ *NbDaun(P)* memberikan banyaknya daun dari pohon *P*.

Pemeriksaan apakah sebuah simpul merupakan simpul daun merupakan persoalan basis 1. Oleh sebab itu, kasus pohon kosong adalah kasus khusus.

Kasus khusus : $\text{NbDaun}(\text{//}\backslash\backslash) = 0$

Jika bukan pohon kosong, gunakan fungsi untuk pohon model-1 (setelah ini) :

NbDaun1(P) }

RepPrefix : PohonBiner \rightarrow List of elemen

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

Basis : $\text{RepPrefix}(\text{//}\backslash\backslash) = []$

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

REALISASI

NbElmt (P) : {boleh model basis-0 }
 if *IsEmpty(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)
{ NbDaun1 terdefinisi pada pohon biner model-1 }

RepPrefix (P) :
 if *IsEmpty(P)* then {Basis 0} []
 else {Rekurens}
 Konkat(Konso(Akar(P), *RepPrefix(Left(P))*), *RepPrefix(Right(P))*)
{ Konkat dan Konso sudah terdefinisi pada List }

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 }

type PohonBiner : <L : PohonBiner, A : elemen, R : PohonBiner> {notasi Infix}, atau

type PohonBiner : <A : elemen, L : PohonBiner, R : PohonBiner> {notasi prefix}, atau

type PohonBiner : <L : PohonBiner, R : PohonBiner, A : elemen> {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 → elemen

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

Left : PohonBiner tidak kosong → PohonBiner

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

Right : PohonBiner tidak kosong → 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

IsOneElmt : PohonBiner tidak kosong → boolean

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

IsUnerLeft : PohonBiner tidak kosong → boolean

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

IsUnerRight : PohonBiner tidak kosong → boolean

{ IsUnerRight(P) true jika P hanya mengandung sub pohon kanan: (// A R\\) }

IsBiner : PohonBiner tidak kosong → boolean

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

IsExistLeft : PohonBiner tidak kosong → boolean

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

IsExistRight : PohonBiner tidak kosong \rightarrow boolean

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

DEFINISI DAN SPESIFIKASI PREDIKAT LAIN

NbElmt : PohonBiner tidak kosong \rightarrow integer ≥ 1

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

Basis : $NbElmt(\text{//A}\backslash\backslash) = 1$

Rekurens : $NbElmt(\text{//L,A,R}\backslash\backslash) = NbElmt(L) + 1 + NbElmt(R)$

$NbElmt(\text{//L,A}\backslash\backslash) = NbElmt(L) + 1$

$NbElmt(\text{//A,R}\backslash\backslash) = 1 + NbElmt(R)$ }

NbDaun1 : PohonBiner \rightarrow integer ≥ 1

{ *Prekondisi : Pohon P tidak kosong* }

{ *NbDaun1(P) memberikan banyaknya daun dari pohon P* :

Basis : $NbDaun1(\text{//A}\backslash\backslash) = 1$

Rekurens : $NbDaun1(\text{//L,A,R}\backslash\backslash) = NbDaun1(L) + NbDaun1(R)$

$NbDaun1(\text{//L,A}\backslash\backslash) = NbDaun1(L)$

$NbDaun1(\text{//A,R}\backslash\backslash) = NbDaun1(R)$ }

RepPrefix: PohonBiner tidak kosong \rightarrow List of elemen

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

Basis : $RepPrefix(\text{//A}\backslash\backslash) = [A]$

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

$RepPrefix(\text{//L,A}\backslash\backslash) = [A] o RepPrefix(L)$

$RepPrefix(\text{//A,R}\backslash\backslash) = [A] o RepPrefix(R)$ }

REALISASI

NbElmt (P) : {P tidak kosong}

```

if IsOneElmt(P) then 1           {Basis}
else depend on P                {Rekurens}
    IsBiner(P)      : NbElmt(Left(P)) + 1 + NbElmt(Right(P))
    IsUnerLeft(P)   : NbElmt(Left(P)) + 1
    IsUnerRight(P)  : 1 + NbElmt(Right(P))

```

NbDaun (P) :

```

if IsOneElmt(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 IsOneElmt(P) then Konso(Akar(P), []) {Basis}
else depend on P                {Rekurens}
    IsBiner(P)      : Konkat(Konso(Akar(P), RepPrefix(Left(P))), 
                                RepPrefix(Right(P)))
    IsUnerLeft(P)   : Konso(Akar(P), RepPrefix(Left(P)))
    IsUnerRight(P)  : Konso(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 menghitung banyaknya operator uner pada sebuah pohon ekspresi aritmatika infix.
3. Buatlah spesifikasi dan definisi dari sebuah fungsi yang memeriksa banyaknya simpul yang ada pada level n.
4. Buatlah realisasi dari spesifikasi fungsional terhadap pohon biner sebagai berikut:

DEFINISI DAN SPESIFIKASI PREDIKAT LAIN

IsMember : PohonBiner, elemen → boolean

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

IsSkewLeft : PohonBiner → boolean

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

IsSkewRight : PohonBiner → boolean

{ *IsSkewRight(P)* mengirimkan true jika P adalah pohon condong kanan }

DEFINISI DAN SPESIFIKASI FUNGSI LAIN

LevelOfX : PohonBiner, elemen → integer

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

AddDaunTerkiri : PohonBiner, elemen → PohonBiner

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

AddDaun : PohonBiner tidak kosong, elemen, elemen, boolean → 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 → <PohonBiner,elemen >

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

DelDaun : PohonBiner tidak kosong, elemen → PohonBiner

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

MakeListDaun : PohonBiner → List of elemen

{ *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 → List of elemen

{ *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 → List of elemen

{ *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 → List of elemen

{ *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 → list of elemen

{ *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 $P = //L\ A\ R\backslash\backslash$ adalah sebuah binary search 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. Realisasinya harus dibuat sebagai latihan.

BSearchX : BinSearchTree, elemen \rightarrow boolean

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

AddX : BinSearchTree, elemen \rightarrow BinSearchTree

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

MakeBinSearchTree : List of elemen \rightarrow BinSearchTree

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

DelBTree : BinSearchTree tidak kosong, elemen \rightarrow BinSearchTree

{ *DelBTree(P,X)* menghasilkan sebuah binary search tree 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:

- perbedaan tinggi sub pohon kiri dengan sub pohon kanan maksimum 1
- 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 elemen, integer \rightarrow BinBalTree

{ *BuildBalanceTree(Ls,n)* menghasilkan sebuah balance tree dengan n node, nilai setiap node berasal dari list Ls. }

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 mengeneralisasi 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: 2 integer \rightarrow integer

{ *SigI(a,b)* adalah fungsi untuk menghitung $\text{Sigma}(i)$ 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 \rightarrow integer

{ *SigI3(a,b)* adalah fungsi untuk menghitung $\text{Sigma}(i^3)$ 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

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

SP8 : 2 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* }

P1 : integer → integer

{ *P1(i)* mengirimkan nilai *i+1* }

P4 : integer → integer

{ *P4(i)* mengirimkan nilai *i+4* }

Cube : integer → integer

{ *Cube(i)* mengirimkan nilai i^3 }

T : integer → real

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

REALISASI

```
Id(i) : i
P1(i) : i+1
P4(i) : i+4
Cube(i) : 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 :

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

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:

DEFINISI DAN SPESIFIKASI

type numerik : union dari type integer dan real

Sigma : integer, integer, (integer → numerik), (integer → numerik) → numerik

{ *Sigma (a,b,f,s)* adalah penjumlahan dari deret/serie $f(i)$, dengan mengambil nilai subseri a , $s(a)$, $s(s(a))$,..., pada interval $[a..b]$ atau 0 jika interval kosong }

REALISASI

```
Sigma(a, b, f, s) : if a > b then {Basis-0}  
                          0  
                          else {Rekurens}  
                          f(a) + Sigma(s(a), b, f, s)
```

Maka :

$\text{Sigma}(a,b,\text{Id},P1)$ identik dengan $\text{SigI}(a,b)$

$\text{Sigma}(a,b,\text{Cube},P1)$ identik dengan $\text{SigI3}(a,b)$

$\text{Sigma}(a,b,\text{T},P4)$ identik dengan $\text{SP8}(a,b)$

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 $\text{max2}(a,b)$ adalah fungsi yang mengirimkan nilai maksimum dari a,b.

Ekspresi tsb. dapat ditulis : $\text{max2}(3,5+x)$

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 : $\lambda x. x+4$

T : $\lambda x. 1/((x+1)*(x+3))$

Aplikasi terhadap ekspresi lambda :

Tuliskan:

$\lambda x. x^3 (2)$ sebagai ganti dari $\text{Cube}(2)$
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 : $\text{Sigma}(a, b, \lambda x. 1/((x+1)+(x+3)), \lambda x. x+4)$

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.

Pasangan-pasangan ekspresi berikut adalah ekivalen :

1. $\lambda x, y. x+y$
 $\lambda x. \lambda y. x+y$
2. $(\lambda x, y. x+y) (2, 3)$
 $(\lambda x. \lambda y. x+y) (2, 3)$
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 2 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: $(\underline{\text{real}} \rightarrow \underline{\text{real}}), \underline{\text{real}} \rightarrow (\underline{\text{real}} \rightarrow \underline{\text{real}})$

DerivTitikX: $((\underline{\text{real}} \rightarrow \underline{\text{real}}), \underline{\text{real}}), \underline{\text{real}} \rightarrow \underline{\text{real}}$

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 nilai $X = 5$ dituliskan sebagai aplikasi dari DerivTitikX dengan parameter:

$(\lambda y. y^3, 0.005) (5)$

adalah nilai $f'(x)$ pada titik $x=5$

DEFINISI DAN SPESIFIKASI

Derivasi : (real → real), real → (real → real)

{ Derivasi (f, dx) adalah derivasi fungsi $f(x)$ dengan interval dx : $(f(x+dx)-f(x))/dx$ }

DerivTitikX : ((real → real), real), real → real

{ DerivTitikX ($(f, dx), NilaiX$) adalah nilai derivasi fungsi $f(x)$ dengan interval dx pada titik X : Aplikasi dari fungsi dengan parameter Derivasi(f, dx) dan NilaiX. Karena DerivTitikX adalah aplikasi terhadap fungsi, maka DerivTitikX($(f, dx), NilaiX$) dapat dituliskan:

- a. Realisasi-1: dengan hanya melakukan aplikasi terhadap Derivasi
- b. Realisasi-2: dengan ekspresi lambda dan akan diinstansiasi dengan NilaiX }

REALISASI-1

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

APLIKASI

```
{ DerivTitikX((f, dx), NilaiX) }  
⇒ Derivasi(f, dx)
```

REALISASI-2 (DENGAN EKSPRESI LAMBDA)

```
Derivasi(f, dx) :  
  ( f(x+dx) - f(x) ) / dx  
DerivTitikX((f, dx), NilaiX) :  
  λx. ( f(x+dx) - f(x) ) / dx (NilaiX)
```

Catatan :

- Terjemahan fungsi di atas tidak mudah, dan sangat spesifik. Lihat Buku II.

Static and dynamic binding

Perhatikan ekspresi sebagai berikut :

```
let n=3 in  
  let f = λx. x+n in  
    let n=2 in f(4)
```

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

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)
DEFINISI DAN SPESIFIKASI	
OffSetList : List of <u>integer</u> tidak kosong, (<u>integer</u> → <u>integer</u>) → List of <u>integer</u> { <i>OffSetList (Li,OffSet)</i> dengan <i>Li</i> adalah list integer dan <i>OffSet</i> adalah sebuah fungsi dengan definisi <i>OffSet(i)</i> melakukan offset terhadap nilai <i>i</i> . <i>OffSetList</i> menghasilkan sebuah list integer dengan elemen yang sudah di-offset }	
REALISASI	
<pre>OffSetList(Li,OffSet) : if IsOneElmt(Li) then [OffSet(FirstElmt(Li))] {basis} else {rekurens} Konso(OffSet(FirstElmt(Li)),OffSetList(Tail(Li),OffSet))</pre>	
BEBERAPA CONTOH OFFSET	
{ <i>f</i> adalah Plus 2 } OffSet ≡ Plus2(i) : i + 2 { <i>f</i> adalah Minus 1 } OffSet ≡ Minus1(i) : 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 (DENGAN EKSPRESI LAMBDA)

```
⇒ OffSetList([1,3,6,0,-9,45], λi. i+2 )
⇒ OffSetList([1,3,6,0,-9,45], λi. i-1 )
⇒ OffSetList([31,1,3,26,0], λi. depend on i
    0 ≤ i ≤ 40 : 10
    41 ≤ i ≤ 60 : 5
    61 ≤ i ≤ 89 : 3
    i > 89 : 1
    else : 0 )
```

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.

FILTERLIST	FilterList(List,f)
DEFINISI DAN SPESIFIKASI	
<p>FilterList : List of <u>integer</u> tidak kosong, (<u>integer</u> → <u>boolean</u>) → List of <u>integer</u> { <i>FilterList (Li,f)</i> dengan <i>Li</i> adalah list of integer dan <i>f</i> adalah sebuah predikat dengan definisi <i>f(i)</i> menghasilkan true jika <i>i</i> memenuhi suatu kondisi tertentu. <i>FilterList</i> menghasilkan sebuah list integer dengan elemen yang memenuhi predikat <i>f</i>. }</p>	
REALISASI	
	<pre>FilterList(Li,f) : <u>if</u> IsOneElmt(Li) <u>then</u> {basis} <u>if</u> f(FirstElmt(Li)) <u>then</u> [FirstElmt(Li)] <u>else</u> [] <u>else</u> {rekurens} <u>if</u> f(FirstElmt(Li)) <u>then</u> Konso(FirstElmt(Li),FilterList(Tail(Li),f)) <u>else</u> FilterList(Tail(Li),f)</pre>

BEBERAPA CONTOH FUNGSI F

```
{ filter adalah integer positif : IsPos? (i) benar jika i positif }
  f ≡ IsPos(i) : i>0
{ filter adalah integer positif : IsNeg? (i) benar jika i negatif }
  f ≡ IsNeg(i) : i<0
{ filter adalah: IsKabisat (i) : bilangan kelipatan 4 tapi bukan kelipatan 100 }
  f ≡ IsKabisat(i) : (i mod 4 = 0) and (i mod 100 ≠ 0)
```

APLIKASI

```
⇒ FilterList([1,3,6,0,-9,45], IsPos)
⇒ FilterList([-1,3,-6,0,-9,45], IsNeg)
⇒ FilterList([31,1,3,26,0], IsKabisat)
```

APLIKASI (DENGAN EKSPRESI LAMBDA)

```
⇒ FilterList([1,3,6,0,-9,45], λi. i>0)
⇒ FilterList([-1,3,-6,0,-9,45], λi. i<0)
⇒ FilterList([31,1,3,26,0], λi. (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.

FilterRekLIST

FilterRekList(List,f)

DEFINISI DAN SPESIFIKASI

FilterRekList : List of list integer tidak kosong, (List of list of integer → List of integer)
→ List of integer

{ *FilterRekList (Si,f)* dengan *Si* adalah list of list integer yang tidak kosong dan *f* adalah sebuah fungsi dengan definisi *f(S)* menghasilkan list of integer dengan atom yang memenuhi suatu aturan tertentu. *FilterRekList* menghasilkan list integer yang dipenuhi oleh *f*. }

REALISASI

```
FilterRekList(Si,f) : f(Si)
```

BEBERAPA CONTOH F

{ *LINEAR(S)* adalah sebuah fungsi yang menerima sebuah list of list integer tidak kosong dan menghasilkan list integer berasal dari atom-atom list *S* }

```
f(S) ≡ LINEAR(S) :  
    if IsOneElmt(S) then  
        if IsAtom(FirstList(S)) then {Basis-1}  
            Konso(FirstList(S), [])  
        else { bukan atom, FirstList(S) harus dilinearkan }  
            LINEAR(FirstList(S)) {Rekurens}  
    else {Rekurens}  
        if IsAtom(FirstList(S)) then  
            Konso(FirstList(S), LINEAR(TailList(S)))  
        else { bukan atom, FirstList(S) harus dilinearkan }  
            Konkat(LINEAR(FirstList(S)), LINEAR(TailList(S)))
```

{ *FilterPos(S)* adalah sebuah fungsi yang menerima sebuah list of list integer dan menghasilkan list integer berasal dari atom-atom list, hanya jika atomnya positif }

```
f(S) ≡ FilterPos(S) :  
    if IsOneElmt(S) then  
        if IsAtom(FirstList(S)) then  
            if FirstList(S)>0 then {Basis-1}  
                Konso(FirstList(S), [])  
            else []  
        else { bukan atom, FirstList(S) harus dilinearkan }  
            FilterPos(FirstList(S)) {Rekurens}  
    else {Rekurens}  
        if IsAtom(FirstList(S)) then  
            if FirstList(S)>0 then  
                Konso(FirstList(S), FilterPos(TailList(S)))  
            else FilterPos(TailList(S))  
        else { bukan atom, FirstList(S) harus dilinearkan }  
            Konkat(FilterPos(FirstList(S)), FilterPos(TailList(S)))
```

APLIKASI (DENGAN EKSPRESI LAMBDA)

⇒

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. Jadi **ekspresi lambda tidak boleh rekursif**.