PENDAHULUAN

Pradigma Pemrograman

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

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

Dalam pemrograman pun ada beberapa paradigma, masing-masing mempunyai 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" memprogram atau memprogram secara coba-coba *("trial and error")*. Program harus dihasilkan dari proses pemahaman permasalahan, analisis, sintesis dan dituangkan menjadi kode dalam bahasa komputer secara sistematis dan metodologis.

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

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

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

Beberapa paradigma dalam pemrograman :

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

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

Algoritma + Struktur Data = Program.

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

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

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

2. Paradigma pemrograman **Fungsional**

Paradigma ini didasari oleh konsep pemetaaan 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 atau 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. Pada kurikulum Jurusan teknik Informatika ITB, pemrograman dengan paradigma ini menjadi bagian dari kuliah Logika Informatika.

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

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

sebuah kelas juga mendapatkan turunan atribut dari kelas di atasnya. Paradigma ini menawarkan konsep modularitas, penggunaan kembali, kemudahan modifikasi.
Dalam paradigma ini, masih terkandung dari paradigma imperatif, karena mengkonstruksi program dari objek dan kelas adalah tidak berbeda dengan mengkonstruksi program dari struktur data dan algoritma, dengan cara enkapsulasi menjadi kelas. Kedekatan antara paradigma ini dengan paradigma lain dapat dilihat dari bahasa-bahasa bukan berorientasi obyek murni, yaitu bahasa prosedural atau fungsional yang ditambahi dengan ciri orientasi objek.

4. **Paradigma Konkuren**
Paradigma ini didasari oleh kenyataan bahwa dalam keadaan nyata, sebuah sistem komputer harus menangani beberapa proses (*task*) yang harus dieksekusi bersama dalam sebuah lingkungan (baik mono atau multi prosesor). Pada pemrograman konkuren, pemrogram tidak lagi berpikir sekuensial, melainkan harus menangani komunikasi dan sinkronisasi antar task. Pada jaman sekarang, aspek konkuren semakin memegang peranan penting dan beberapa bahasa menyediakan mekanisme dan fasilitas yang mempermudah implementasi program konkuren.
Salah satu aspek penting dalam pemrograman berorientasi objek dan merupakan topik pemrograman yang "lanjut" adalah bagaimana menangani objek yang "hidup bersama, secara konkuren". Maka pada kurikulum Jurusan Informatika, paradigma pemrograman konkuren dan berorientasi Objek dicakup dan disatukan dalam matakuliah Pemrograman Objek Konkuren.

Bahasa Pemrograman

Berbicara mengenai bahasa pemrograman, ada banyak sekali bahasa pemrograman, mulai dari bahasa tingkat rendah (bahasa mesin dalam biner), bahasa asembler (dalam kode mnemonik), bahasa tingkat tinggi, sampai bahasa generasi ke empat (4GL).
Bahasa Pemrograman berkembang dengan cepat sejak tahun enampuluhan, seringkali dianalogikan dengan menara Babel yang berakibat manusia menjadi tidak lagi saling mengerti bahasa masing-masing. Untuk setiap paradigma, tersedia bahasa pemrograman yang mempermudah implementasi rancangan penyelesaian masalahnya. Contoh bahasa-bahasa pemrograman yang ada :
1. Prosedural : Algol, Pascal, Fortran, Basic, Cobol, C , Ada
2. Fungsional : LOGO, APL, LISP
3. Deklaratif : Prolog
4. Object oriented murni: Smalltalk, 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 Memprogram Tidak Sama Dengan Belajar Bahasa Pemrograman

Belajar memprogram adalah belajar tentang strategi pemecahan masalah, metodologi dan sistematika pemecahan masalah tersebut kemudian menuangkannya dalam suatu notasi yang disepakati bersama. Beberapa masalah akan cocok kalau diselesaikan dengan suatu paradigma tertentu. Karena itu, pengetahuan tentang kelas persoalan penting adanya.
Pada hakekatnya, penggunaan komputer untuk memecahkan persoalan adalah untuk tidak mengulang-ulang kembali hal yang sama. Strategi pengenalan masalah melalui

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

Belajar memprogram lebih bersifat pemahaman persoalan, analisis, sintesis.

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

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

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

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

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

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

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

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

Tujuan Kuliah Pemrograman Fungsional

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

Secara lebih spesifik, mahasiswa diharapkan mampu untuk :

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

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

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

Ikhtisar Diktat

Diktat ini terdiri dari dua bagian

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

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

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

Buku II – LISP difokuskan pada penulisan program fungsional dalam bahasa LISP. Selain membahas LISP secara ringkas dan pola translasi dari notasi fungsional ke LISP, pada bagian ini semua konsep yang ditulis di bagian pertama akan diberikan aturan translasinya. Pada bagian kedua ini juga dicakup aspek eksekusi program LISP: mahasiswa belajar memahami aspek eksekusi (evaluasi fungsional) melalui contoh-contoh yang diberikan, yang sudah dicoba dengan kompilator CGLISP. Menyadari bahwa dalam keluarga bahasa LISP banyak "varian" nya, maka diusahakan, agar instruksi yang dipakai hanyalah instruksi standard.
Beberapa fungsi yang dituliskan dalam bagian pertama sengaja diulang penulisannya, untuk mempermudah pembaca dalam membaca. Tidak semua fungsi dasar dan contoh yang diberikan pada bagian kesatu diterjemahkan dalam bagian kedua. Hal ini sengaja dilakukan agar mahasiswa berlatih sendiri dengan mesin untuk menterjemahkannya, karena sekarang mahasiswa sudah mempunyai mesin pengeksekusi..

Daftar Pustaka :

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

NOTASI FUNGSIONAL

Sebuah program komputer adalah “model”, yang mewakili solusi persoalan tertentu dalam informatik yang akan diselesaikan dengan komputer. Program berisi kumpulan informasi penting yang mewakili persoalan itu.
Pada paradigma pemrograman fungsional solusi persoalan diungkapkan menjadi identifikasi dari satu atau beberapa fungsi, yang jika di"aplikasi" dengan nilai yang diberikan akan memberikan hasil yang diharapkan. Dalam paradigma fungsional, program direpresentasi dalam: himpunan nilai type, dengan nilai-nilai dari type adalah konstanta.

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

- type dasar
- type terkomposisi (bentukan)

Untuk menuliskan suatu program fungsional, dipakai suatu bahasa ekspresi .
Ada tiga macam bentuk komposisi ekspresi :

- ekspresi fungsional dasar
- konditional
- rekursif

Masing-masing ekspresi akan dibahas pada bagian selanjutnya.

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

Perhatikan sebuah program yang ditulis dalam bahasa algoritmik sebagai berikut :

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

APLIKASI

$\Rightarrow$ 3+4

$\Rightarrow$ 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 *topdow*n adalah strukturasi teks. Sebuah fungsi mewakili sebuah tingkatan abstraksi. Dengan mengenalkan (mendefinisikan) sebuah fungsi, maka pemrogram memperkaya "khasanah", perbendaharaan dari fungsi yang tersedia, dan karena sudah didaftarkan maka dapat digunakan kemudian.

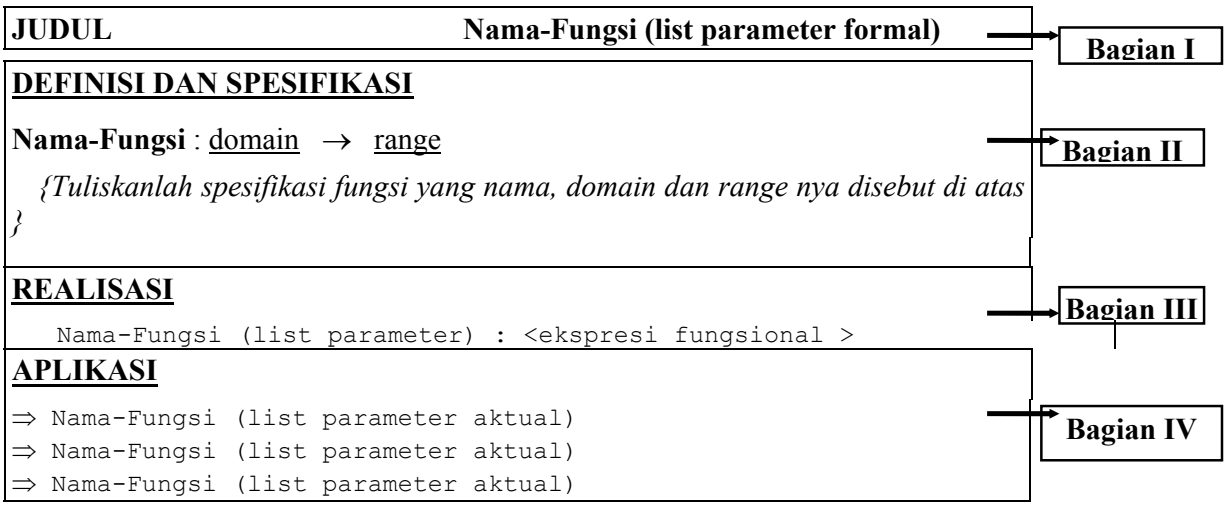
Konstruksi program fungsional : definisi-spesifikasi-realisasi-aplikasi

Tahapan	Deskripsi
Definisi	Kata kuncinya adalah **memberikan identitas fungsi, yaitu m**enentukan **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 deskripsi sangat ringkas dari fungsi dan nama serta parameter formalnya. Bagian ini cukup memberikan penjelasan bagi pemakai fungsi andaikata sudah pernah direalisasikan.
Bagian kedua berisi definisi dan spesifikasi fungsi sesuai dengan keterangan di atas.; kedua ini (definisi dan spesifikasi) tidak dipisahkan satu sama lain karena sangat erat kaitannya.
Bagian ketiga berisi realisasi fungsi, yaitu **ekspresi fungsional** yang ditulis untuk mencapai spesifikasi yang dimaksudkan. Sebuah definisi dan spesifikasi yang sama dapat direalisasikan dalam beberapa ekspresi ! Perhatikanlah pula bahwa dalam realisasi fungsi, nama fungsi dituliskan berikut ekspresinya. Tidak diperlukan kata kunci "fungsi" karena dalam konteks fungsional, semua objek adalah fungsi.
Bagian keempat berisi contoh aplikasi fungsi, jika dipandang perlu, dan bahkan hasilnya. Bagian inilah yang sebenarnya akan merupakan interaksi langsung dengan pemakai pada konsteks eksekusi.
Bagian teks yang berupa komentar dituliskan di antara kurung kurawal
Notasi fungsional ini dibuat untuk menuliskan rancangan program supaya lebih mudah dibaca oleh "manusia".

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



Lihatlah contoh dari penggunaan notasi ini pada bagian berikutnya Seluruh sisa buku ini akan memakai notasi seperti di atas.
Notasi fungsional ini diadopsi dari [3] dan dikembangkan khusus untuk perkuliahan ini, dan mampu menampung semua konsep pemrograman fungsional, walaupun tidak mungkin dieksekusi (karena tidak mempunyai pemroses bahasa). Kekuatan aspek konseptual, statis dan aspek desain ini justru diperoleh karena 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, maka notasi ini harus dilengkapi dengan aturan translasi ke bahasa riil yang ada. Buku kedua akan melengkapi notasi ini dengan aturan translasi ke LISP.

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

EKSPRESI DASAR

Seperti telah dijelaskan pada bagian pendahuluan, pada pemrograman fungsional, pemrogram mulai dari **fungsi dasar** yang disediakan oleh pemroses bahasa untuk membuat fungsi lain yang melakukan aplikasi terhadap fungsi dasar tersebut.
Fungsi yang paling dasar pada program fungsional disebut **operator**. Pada ekspresi fungsional, sebagai "titik awal", dinyatakan bahwa tersedia operator aritmatika, operator relasional dan operator boolean.

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

Ekspresi adalah sebuah teks yang terdiri dari: nama, simbol, operator, fungsi, ( ), yang dapat menghasilkan suatu nilai berkat evaluasi dari ekspresi.
Ekspresi aritmatika (numerik), adalah ekspresi dengan operator aritmatika '*', '/', '+, '-' dapat dituliskan dalam bentuk infix, prefix atau postfix. Pada notasi fungsional, jenis ekspresi yang dipakai adalah Infix, karena lebih manusiawi.

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

Ekspresi di atas tidak mengandung nama, melainkan hanya mengandung simbol angka numerik, operator dan tanda kurung
Hasil evaluasi (perhitungan) suatu ekspresi dasar dapat berupa nilai numerik atau boolean. Ekspresi yang hasilnya numerik disebut ekspresi numerik (aritmatika). Ekspresi yang hasilnya boolean disebut ekspresi boolean. Jika sebuah ekspresi boolean operatornya numerik, disebut pula ekspresi relasional.

Selain menggunakan konstanta numerik tersebut, ekspresi dapat berupa ekspresi aljabar :
- abstraksi dengan menggunakan "nama"
- nama mempunyai "nilai"

yang hanya mengandung operator aritmatika.
Evaluasi ekspresi tergantung kepada presedensi (prioritas evaluasi) dan aturan yang ditetapkan.
Contoh :

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

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

Berikut ini diberikan contoh-contoh persoalan yang dapat diselesaikan secara fungsional hanya dengan membuat ekspresi dasar fungsional, yaitu dengan operator aritmatika, operator relasional, operator boolean dan fungsi dasar terhadap bilangan seperti mod, div, abs.
Pada hakekatnya, fungsi akan menghasilkan suatu nilai karena di dalam fungsi tsb dilakukan evaluasi ekspresi.

Contoh-1 Ekspresi numerik : PANGKAT DUA

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

PANGKAT2	**FX2(x)**

DEFINISI DAN SPESIFIKASI

FX2 : integer → integer

{FX2 (x) menghitung pangkat dua dari x, sebuah bilangan integer }

REALISASI

```
FX2 (x) : x * x
```

APLIKASI

```
⇒ FX2(1)
⇒ FX2(0)
⇒ FX2(-1)
```

Contoh-2 Ekspresi numerik: PANGKAT TIGA

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

PANGKAT3 (versi-1)	**FX3(x)**

DEFINISI DAN SPESIFIKASI

FX3 : integer → integer

{FX3 (x) menghitung pangkat tiga dari x, sebuah bilangan integer }

REALISASI

```
FX3 (x) : x * x * x
```

APLIKASI

```
⇒ FX3(1)
⇒ FX3(8)
⇒ FX3(-1)
```

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

Persoalan :

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

PANGKAT3 (versi 2) **FX3(x)**

DEFINISI DAN SPESIFIKASI

FX3 : integer → integer

{FX3 (x) menghitung pangkat tiga a dari x, sebuah bilangan integer, dengan aplikasi FX2 sebagai fungsi antara}

FX2 : integer → integer

{FX2 (x) menghitung pangkat dua dari x, sebuah bilangan integer }

REALISASI

```
FX3 (x) : x * FX2(x)
```

APLIKASI

```
⇒ FX3(8)
⇒ FX3(1)
⇒ FX3(-1)
```

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

Persoalan :

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

MAKSIMUM 3 BILANGAN INTEGER **MAX3(a,b,c)**

DEFINISI DAN SPESIFIKASI

MAX3 : 3 integer → integer

{MAX3 (a,b,c) menentukan nilai maksimum dari 3 bilangan integer a,b, dan c}

MAX2 : 2 integer → integer

{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

a.Definisi : menentukan nama fungsi, domain dan range

MO : 4 bilangan bulat >0 → real { Nama fungsi : MO }
{ Domain : 4 buah bilangan bulat }
{ Range : bilangan riil }

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

b. 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 = 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 ke tiga, 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

MO : 4 integer ≥ 0 → real

{ MO (u,v,w,x): menghitung rata-rata dari dua buah bilangan integer, yang bukan maksimum dan bukan minimum dari 4 buah integer: (u+v+w+x-min4 (u,v,w,x)-max4 (u,v,w,x))/2

max4 : 4 integer >0 → integer

{max4 (i,j,k,l) menentukan maksimum dari 4 buah bilangan integer}

min4 : 4 integer > 0 → integer

{min4 (i,j,k,l) menentukan minimum dari 4 buah bilangan integer}

max2 : 2 integer >0 → integer

{max2 (a,b) menentukan maksimum dari 2 bilangan intege, hanya dengan ekspresi aritmatika: jumlah dari kedua bilangan ditambah denganselisih kedua bilangan, hasilnya dibagi 2}

min2 : 2 integer > 0 → integer

{min2 (a,b) menentukan minimum dari 2 bilangan integer, , hanya dengan ekspresi aritmatika :jumlah dari kedua bilangan – selisih kedua bilangan , hasilnya dibagi 2}

REALISASI

```
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
```

APLIKASI

```
⇒ 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)**

DEFINISI DAN SPESIFIKASI

IsPositif? : integer → boolean

{IsPositif? (x) benar jika x positif }

REALISASI

```
IsPositif?(x) : x ≥ 0
```

APLIKASI

```
⇒ IsPositif?(1)
⇒ IsPositif? (0)
⇒ IsPositif? (-1)
```

Contoh-6 Ekspresi boolean : APAKAH HURUF A

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

APAKAH HURUF A	**IsAnA?(C)**

DEFINISI DAN SPESIFIKASI

IsAnA : character → boolean

{IsAnA (C) benar jika c adalah karakter (huruf) 'A' }

REALISASI

```
IsAnA?(c) : c = 'A'
```

APLIKASI

```
⇒ IsAnA?(1)
⇒ IsAnA? (0)
⇒ IsAnA? (-1)
```

Contoh-7 Ekspresi boolean : APAKAH ORIGIN

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

APAKAH ORIGIN	**IsOrigin?(x,y)**

DEFINISI DAN SPESIFIKASI

IsOrigin? : real, real → boolean

{IsOrigin? (x,y) benar jika (x,y) adalah dua nilai yang mewakili titik origin (0,0) }

REALISASI

```
IsOrigin?(x,y) : x=0 and y=0
```

APLIKASI

```
⇒ IsOrigin?(1,0)
⇒ IsOrigin? (1,1)
⇒ IsOrigin? (0,0)
```

Contoh-7 Ekspresi boolean dengan operator boolean: APAKAH VALID

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

APAKAH VALID	**IsValid?(x)**

DEFINISI DAN SPESIFIKASI

IsValid? : integer → boolean

{IsValid? (x) benar jika (x) bernilai lebih kecil 5 atau lebih besar dari 500 }

REALISASI

```
IsValid?(x) : x <5 or x>500
```

APLIKASI

```
⇒ IsValid?(5)
⇒ IsValid? (0)
⇒ IsValid? (500)
⇒ IsValid? (6000)
```

Evaluasi Fungsi

Evaluasi fungsi :

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

* 2 3

yang dapat kita pandang sebagai "fungsi" *(2,3) dalam notasi fungsional.
Evaluasi ekspresi dalam konteks fungsional adalah melakukan aplikasi fungsi sambil melakukan evaluasi dari ekspresi yang mengandung operan. Karena sebuah ekspresi dapat mengandung lebih dari satu operan dan aplikasi fungsi, maka urutan dari evaluasi dapat bermacam-macam dan prioritas dari operan menentukan urutan evaluasi. Untuk ketepatan evaluasi dari ekspresi yang mengandung operan, disarankan untuk menuliskan tanda kurung secara eksplisit. Untuk evaluasi yang dilakukan berdasarkan aplikasi fungsi, kita harus mempelajari aturan evaluasi dari bahasa yang bersangkutan.

Contoh-1 Evaluasi fungsi :

Persoalan :
Buatlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menrima 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 real → real

{LS(x1,x2,y1,y2) adalah jarak antara dua buah titik (x1,x2) dengan (y1,y2) }

DEFINISI DAN SPESIFIKASI FUNGSI ANTARA

dif2 : 2 real → real

{dif(x,y) adalah kuadrat dari selisih antara x dan y }

FX2 : real → real

{ FX2 (x) adalah hasil kuadrat dari x }

REALISASI

```
FX2 (x)   : x * x
dif2 (x,y) : FX2(x - y)
LS (X1,y1,x2,y2):V dif2(y2,y1) + dif2(x2,x1)
```

Berikut ini diberikan contoh perhitungan yang dilakukan oleh "pemroses bahasa" pada saat aplikasi, yaitu jika evaluasi dilakukan darikiri ke kanan.

Evaluasi LS (1,3,5,6)

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

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

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

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

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

--> V 9 + FX2(5-1)                  {ekspansi dif(5,1)}

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

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

--> V 9 + 16                        {reduksi *}

--> V 25                            {reduksi +}

--> 5                               {reduksi V }
```

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

Suatu ekspresi "antara", yaitu ekspresi yang dituliskan untuk sementara di dalam fungsi namun tidak dikomunikasikan ke dunia luar fungsi tersebut, dapat diberi nama (yang bukan merupakan nama fungsi). Nama ini hanya berlaku sementara di "dalam" sebuah fungsi.
Pemakaian fungsi atau nama antara tergantung pada generalitas solusi yang kita inginkan. Fungsi selalu dapat digunakan dalam konteks lain.

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

Bentuk umum

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

dengan :

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

Bentuk lebih Umum :

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

              <Realisasi-Fungsi>
```

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

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

Pemakaian let

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

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

Menghindari evaluasi yang berulang
Misalnya dalam ekspresi (1+a*b) * (1-2*a*b) harus dilakukan evaluasi
a * b sebanyak dua kali
Untuk menghindari evaluasi berulang dapat ditulis :

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

Memudahkan interpretasi/pembacaan teks program :
Misalnya pada MO (u,v,w,x)

REALISASI

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

REALISASI

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

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

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

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

Dalam hal ini, Konteks/scope harus diperhatikan
Perhatikan Contoh berikut yang "membingungkan" :

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

Sebaiknya semua nama lokal tidak sama dengan nama parameter formal.