Draft Diktat Kuliah
Dasar Pemrograman
(Bagian Pemrograman Prosedural)

Oleh :

Inggriani Liem

Kelompok Keahlian

Rekayasa Perangkat Lunak dan Data

STEI - ITB

Edisi April 2007

Halaman ini sengaja dibiarkan kosong

PRAKATA

Diktat ini disusun secara khusus untuk keperluan pengajaran kuliah Algoritma dan Pemrograman di lingkungan Departemen Teknik Informatika ITB. Diktat ini merupakan revisi dari diktat yang pernah disusun untuk perkuliahan algoritma dan Pemrograman sejak tahun 1990. Diktat ini tidak dapat dipisahkan dari Seri "Catatan Singkat" dan "Program Kecil", yaitu tulisan ringkas mengenai pemrograman dalam bahasa Pascal, C, dan Ada yang dipakai secara paralel. Diktat ini juga sangat erat hubungannya dengan matakuliah terkait yaitu diktat "Pemrograman Fungsional", diktat "Struktur Data", dan diktat "Pemrograman Berorentasi Objek".

Pada matakuliah Algoritma dan Pemrograman bagian Prosedural, mahasiswa dibekali dengan metodologi pemrograman prosedural, dengan notasi algoritmik yang terstruktur serta implementasinya dalam bahasa tingkat tinggi prosedural.

Mahasiswa dianjurkan untuk menuliskan solusi mereka sebelum membaca solusi pada diktat ini, dan kemudian segera menerjemahkan solusi algoritmik pada diktat ini menjadi program yang dapat dieksekusi mesin dalam salah satu bahasa tingkat tinggi yang diajarkan. Pertanyaan-pertanyaan yang sengaja tidak dijawab pada beberapa solusi dimaksudkan untuk didiskusikan di luar kuliah. Biasanya pertanyaan-pertanyaan tersebut mengandung ide pedagogik yang jawabannya perlu mendapatkan kupasan yang matang dari pengajar.

Bagi pengajar yang ingin memanfaatkan buku ini, tersedia buku lain yang merupakan buku pegangan mengajar dan perancangan dari perkuliahan secara keseluruhan.

Dikat ini dapat diwujudkan berkat diktat pengajaran IF221 (kurikulum lama ITB) yang pernah ditulis pada tahun 1981, kemudian diperbarui pada tahun 1996 dan pengajaran algoritmik yang pernah diberikan oleh staf pengajar laboratorium pemrograman di Universitas Grenoble I - Perancis (Prof. Scholl, Peyrin dan Cagnat).

Terima kasih kepada :
- Anita Burhan, asisten dosen yang sudah melakukan konversi dari file lama menjadi yang terbaca di lingkungan DOS pada tahun 1990.
- Mahasiswa IF221 Semester I-90/91yang telah mengoreksi draft dan cikal bakal diktat ini, telah memberikan banyak inspirasi untuk perbaikan pengajaran. Demikian pula mahasiswa Semester I - 91/92 dan 92/93
- Ruth Kurniawati, asisten perkuliahan semester I - 91/92 dan Semester I - 92/93, yang telah membantu dengan penuh semangat untuk mengkoreksi dan mengedit versi awal buku ini, serta melengkapi dan men-test program-program terkait.
- Fazat Nur Azizah, salah satu staf pengajar IF1282 yang telah bersedia mencermati ulang sebelum diktat ini diterbitkan untuk mahasiswa TPB-STEI angkatan 2006.

Kritik, saran dan koreksi sangat diharapkan untuk perbaikan diktat ini pada cetakan yang akan datang. Kesalahan ketik yang mengakibatkan kesalahan algoritmik pada diktat ini tak mungkin dikoreksi oleh kompilator.

Bandung, tahun ajaran 2004/2005

Halaman ini sengaja dibiarkan kosong

DAFTAR ISI

PENDAHULUAN

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 Pemrograman

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, karena itu fanatisme terhadap sebuah paradigma, mempersempit wawasan dan kadang berbahaya.

Dalam pemrograman pun ada beberapa paradigma, masing-masing mempunyai priotas strategi analisis 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. Mngharuskan 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, tentu saja tidak mungkin semua paradigma disatukan dalam sebuah mata kuliah. Mahasiswa akan mulai belajar dengan paradigma prosedural.

Beberapa paradigma dalam pemrograman :

1. Paradigma Prosedural atau 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:

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.

2. Paradigma Fungsional

Paradigma fungsional 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.

Paradigma fungsional tidak lagi mempermasalahkan 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 Deklaratif, Predikatif, atau Lojik

Paradigma ini didasari oleh pendefinisian **relasi** antar individu yang dinyatakan sebagai **predikat**. Sebuah program lojik adalah kumpulan aksioma (fakta dan aturan deduksi).

Pada paradigma ini, pemrogram menguraikan 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 Berorientasi Objek (*Object Oriented*)

Paradigma ini didasari oleh **objek.** Sebuah 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 adalah objek mempunyai atribut yang sama dan diturunkan ke semua pbjek yang berada dalam kelas yang sama. Kelas-kelas mempunyai hirarki, anggota dari sebuah kelas juga mendapatkan turunan atribut dari kelas di atasnya. Paradigma ini menawarkan konsep *class*, *generic*, *inheritance*, *polymorphism*, dan menekankan pentingnya pendefinisian statik kelas untuk melahirkan (menciptakan) objek pada saat *runtime*, yang kemudian dimanipulasi atau saling berinteraksi. Definisi kelas memungkinkan adanya penurunan kelas dengan objek pada saat *runtime* yang dapat "berubah" bentuk dengan kelakuan yang disesuaikan.

Namun demikian, literatur pada struktur kontrol mikro untuk mendeskripsikan kelakuan, masih terkandung paradigma imperatif, dengan adanya pernyataan sekuensial, *assignment*, analisis kondisional, dan pengulangan. Namun demikian, mengonstruksi program dari objek dan kelas adalah berbeda dengan mengonstruksi program dari struktur data dan algoritma. Kedekatan antara paradigma ini dengan paradigma lain dapat dilihat dari bahasa-bahasa bukan berorientasi objek murni, yaitu bahasa prosedural atau fungsional yang ditambahi dengan ciri orientasi objek.

Selain keempat paradigma di atas, dalam literatur masih sering disebutkan paradigma yang lain, misalnya:
- Paradigma konkuren, yang erat hubungannya dengan arsitektur perangkat keras yang memungkinkan pemrosesan secara paralel atau perangkat lunak sistem terdistribusi yang mengelola akses konkuren.
- Paradigma relasional, yang didasari entity dan relasi, dan pemrograman dalam bahasa *query* yang memungkinkan diperolehnya suatu himpunan nilai.

Bahasa Pemrograman

Ada banyak sekali bahasa pemrograman, mulai dari bahasa tingkat rendah (bahasa mesin dalam biner), bahasa *assembler* (dalam kode mnemonik), bahasa tingkat tinggi, sampai bahasa generasi 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.
2. Fungsional: LOGO, APL, LISP.
3. Deklaratif/lojik: Prolog.
4. *Object oriented* murni: Smalltalk, Eifel, Jaca, C++.
5. Konkuren: OCCAM, Ada, Java.
6. Relasional: SQL pada basisdata relasional.

Paradigma berorientasi objek mulai ditambahkan pada bahasa-bahasa yang ada. Pemroses Bahasa Pascal dan C versi terbaru dilengkapi dengan fasilitas berorientasi objek, misalnya Turbo Pascal (mulai versi 5.5) pada komputer pribadi (PC) dan C++. Ada beberapa versi LISP dan Prolog yang juga memasukkan aspek *object oriented* (OO).

Suatu program dalam bahasa pemrograman tertentu akan diproses oleh pemroses bahasanya. Ada dua kategori pemroses bahasa, yaitu kompilator dan interpreter.
Dalam melakukan implementasi program, tersedia bahasa pemrograman visual atau tekstual.

Belajar Pemrograman 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 rutin-rutin yang ditulis orang lain. Memang ada algoritma baru yang lahir, tapi relatif lebih sedikit dibandingkan dengan bongkar pasang program yang sudah ada.

Belajar memrogram lebih bersifat pemahaman persoalan, analisis, sintesis. Belajar bahasa pemrograman adalah belajar memakai suatu bahasa, aturan sintaks (tata bahasa), setiap instruksi yang ada, dan tata cara pengoperasian kompilator 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 bahasa pemrograman lebih bersifat keterampilan daripada 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.

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 mengeksekusi program, belumlah lengkap. Sebaliknya, hanya mencetak pemrogram yang langsung "memainkan *keyboard*", mengetik program dan mengeksekusi tanpa analisis dan spesifikasi yang dapat dipertanggung jawabkan juga bukan merupakan praktek yang “baik” (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 apa pun adalah melalui contoh. Seorang yang sedang belajar harus belajar melalui contoh nyata. Berkat contoh nyata itu dia melihat, mengalami dan melakukan pula. 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. Contoh tipikal tersebut dikembangkan dan dipakai untuk belajar dan mengajar sesuai dengan paradigma pemrograman yang diajarkan.

Program: Produk Versus Proses

Beberapa kalangan berpendapat bahwa yang penting dalam sebuah pengembangan program adalah produk. Pendapat ini sudah kuno. Sebuah produk yang baik, mungkin dihasilkan oleh sebuah proses yang “kurang baik” atau bahkan “sangat buruk” karena hasil akhir yang dipolers di sana sini secara tambal sulam. Sebaliknya, proses yang baik pasti dapat menjamin kehadiran suatu produk yang baik.

Aspek penekanan pada **proses** ini sangat ditegaskan dalam kuliah pemrograman (dan nantinya dalam rekayasa perangkat lunak dengan skala lebih besar). Memang pemantauan produk jauh lebih gampang daripada pemantauan proses. Apalagi dengan jumlah mahasiswa yang cukup banyak maka aspek proses seringkali sulit dipantau. Diharapkan bahwa mahasiswa tidak hanya mampu untuk menghasilkan produk program, melainkan juga dapat menjalankan proses pembuatan program seperti yang “sebaiknya” sesuai dengan prosedur yang standard. Sikap mahasiswa yang mau disiplin secara mandiri sangat diharapkan dalam mencapai tujuan kuliah ini.

Program Skala Kecil dan Program Skala Besar

Pada kehidupan nyata, perangkat lunak yang dibutuhkan “berukuran” besar, bahkan sangat besar dan mempunyai persyaratan tertentu. Pembangunan program berskala

besar tidak dapat dilakukan dengan cara yang identik dengan program skala kecil. Ini dapat dianalogikan dengan pembangunan sebuah gedung pencakar langit dibandingkan dengan pembangunan sebuah rumah kecil sederhana.

Sesuatu yang "besar" biasanya terdiri dari komponen-komponen "kecil". Misalnya sebuah peralatan elektronik yang canggih, di dalamnya terdiri dari komponen elektronik "standard" yang dirakit. Ada produsen komponen, dan ada perakit yang memakai komponen. Sebuah gedung besar mempunyai komponen seperti pintu, jendela, ubin yang dapat difabrikasi secara terpisah sebelum dipasang. Sebuah program yang besar dan "baik" biasanya terdiri dari banyak sekali modul/komponen "kecil" yang dikerjakan oleh banyak orang. Pendekatan "merakit" ini juga dilakukan dalam pembangunan perangkat lunak. Maka dapat dikatakan bahwa ada dua kategori pemrogram: penyedia modul/komponen dan pemakai modul/komponen. Mahasiswa Informatika selayaknya mampu untuk menjadi penyedia modul/komponen.

Skala komponen juga makin berkembang, dari komponen setara "suku cadang" dalam industri perakitan mobil, sampai komponen utama yang menjadi dasar, bahkan "*engine*" dari produk secara keseluruhan (munculnya istilah *library, framework, platform*, dan sebagainya).

Pada kuliah-kuliah pemrograman yang mendasar ("Algoritma dan Pemrograman", "Struktur Data", "Pemrograman Berorientasi Objek") di Program Studi Informatika ITB, hanya dicakup pembangunan program-program skala "kecil" (karena memang "kecil"), atau pengembangan modul-modul "kecil" yang merupakan modul/komponen primitif yang nantinya akan merupakan bagian dari sebuah perangkat lunak berskala besar. Sedangkan pembangunan perangkat lunak skala besar akan dicakup dalam modul-modul kuliah kelompok "Rekayasa Perangkat Lunak".

Pemrogram Individu dan Pemrogram dalam Tim

Zaman dulu, tidak jarang ditemui programer "eksentrik", suka menyendiri, jenius yang selalu hanya berhadapan dengan komputer dan menghasilkan kode-kode program "misterius" yang hanya dimengerti olehnya sendiri (Baca tulisan Hoare mengenai *"Programming: Sorcery or Science"* [Hoare-84]).

Zaman itu sudah berlalu. Skala program yang harus dibuat di masa kini sudah jauh lebih besar ukurannya dibandingkan dengan jaman dulu. Ini sudah dijelaskan pada bagian program skala besar dan program skala kecil. Akibatnya, pemrogram diharapkan untuk dapat bekerja dalam tim. Jika dianalogikan dengan dunia musik, maka pemrograman adalah sebuah orkestra, yang harus dipimpin seorang konduktor supaya pertunjukan dapat berjalan dengan baik. Sebuah orkestra selain membutuhkan kerja keras individu, juga membutuhkan kemauan setiap individu untuk bermain selaras sesuai dengan arahan konduktor.

Bekerja dalam tim membutuhkan ***standard*** atau pola kerja yang sama dan aturan main agar ada koodinasi dan tim dapat menghasilkan suatu produk sesuai dengan prosedur. Ada banyak *standard* yang dipakai di dunia pemrograman. Di kelas pemrograman di lingkungan Departemen Teknik Informatika ITB, akan ditetapkan suatu *standard*

pemrograman tertentu yang diharapkan akan dipakai selama kuliah. Aspek ini perlu diperhatikan dan dengan disepakati bersama.

Aspek bekerja dalam tim dan mengikuti *standard* ini akan dicakup dalam perkuliahan di pemrograman Departemen Teknik Informatika ITB. Jadi, salah satu tujuan pengajaran adalah membentuk pemrogram atau perancang program yang harus mampu bekerja sama dengan orang lain.

Kuliah "Algoritma dan Pemrograman" lebih diarahkan kepada *programming*, daripada *coding*.

Aktifitas Mahasiswa

Beberapa kalangan berpendapat **bahwa memprogram adalah menghasilkan program**. Ini dapat dianalogikan dengan melihat seorang penyanyi. Seorang penyanyi tugasnya adalah hanya bernyanyi. Atau seorang olahragawan yang fokusnya adalah bertanding di lapangan untuk cabang olahraga tertentu. Kalau diperhatikan, seorang penyanyi yang baik tidak hanya berlatih untuk menyanyi, namun harus melakukan secara disiplin banyak latihan yang lain, misalnya: latihan pernafasan, menguasai beberapa alat musik, menjiwai suatu peran ketika menyanyi di panggung, dan sebagainya. Seorang olahragawan harus melakukan latihan-latihan rutin, misalnya lari pagi, angkat barbel, senam atau olah raga lain yang menunjang olahraga utamanya. Demikian pula dengan "pemrogram".

Aktivitas mahasiswa yang perlu dilakukan mahasiswa yang sedang belajar memrogram dapat digolongkan dalam kelompok sebagai berikut:

- **Simulasi,** sensibilitas terhadap masalah dan kemungkinan solusi. Kegiatan di kelas: melalui permainan, misalnya mengurut dokumen atau kartu. Permainan di kelas dapat dilakukan jika jumlah siswa sedikit, sulit dilakukan jika jumlah mahasiswa banyak. Permainan dapat disimulasi dengan komputer, sehingga siswa dapat bermain secara mandiri.

- **Analisis kebutuhan *(requirement)* dan desain** masalah secara lebih formal dan membuat dokumen spesifikasi dan rancangan algoritma dalam notasi yang ditetapkan. Mahasiswa harus menuliskan solusi algoritmiknya dalam notasi dan standar yang diberikan. Notasi yang dipakai adalah untuk mengarahkan siswa pada kerangka pemecahan permasalahan, membebaskan siswa dari detil bentuk-bentuk detil aturan penulisan. Dengan notasi pada diktat ini, beberapa kesulitan spesifik dalam pemrograman (misalnya "*loop*" dan "*if*" yang saling melingkupi) dan beberapa konsep pemrograman yang tidak diterapkan lewat instruksi bahasa pemrograman partikular karena tidak tersedia dapat dihindari.

- **Menulis program**, yaitu menerjemahkan algoritma ke program secara "setia" menurut aturan penerjemahan yang diberikan di kelas.

- ***Debugging* dan menguji coba** program, dilakukan mahasiswa secara mandiri dengan komputer. Idealnya dua kali "*run*" sudah cukup untuk mendapatkan program benar: sekali untuk membersihkan program dari salah sintaks, dan kedua

untuk mendapatkan program benar. Kesalahan lojik jarang terjadi kalau analisis benar.

- Mengamati peristiwa **eksekusi**, perlu untuk meningkatkan kepercayaan bahwa jika analisis benar maka sisa pekerjaan menjadi mudah. Pada pemrograman prosedural, aspek ini penting untuk memahami fenomena eksekusi dan perubahan nilai suatu struktur data. Pada paradigma pemrograman lain, aspek ini penting untuk menunjukkan mekanisme eksekusi paradigma yang bersangkutan pada mesin Von Newmann, dan untuk menunjukkan mekanisme kerja pemroses bahasa yang bersangkutan : pada pemrograman fungsional, penting untuk menunjukkan mekanisme aplikasi fungsi, pada pemrograman deklaratif untuk menunjukkan bagaimana proses deduksi dan inferensi dilakukan, pada *object oriented programming* (OOP) untuk menunjukkan bagaimana kaitan suatu objek dengan objek lain terjadi saat *runtime* dan bagaimana beberapa mekanisme seperti *"encapsulation", "inheritance", "memory management"* dilakukan oleh bahasanya.

- **Membaca program**: orang akan dapat menulis dengan baik kalau sering membaca. Hal ini juga berlaku dalam memrogram. Kegiatan yang dapat dilakukan di kelas adalah dengan saling tukar menukar teks algoritma dan saling mengkritik algoritma teman. Mahasiswa harus berlatih sendiri pada kegiatan belajar bersama.

- **Membuktikan kebenaran program secara formal**, satu-satunya hal yang menjamin kebenaran, tetapi kontradiktif dan sulit diterapkan dalam kehidupan sehari-hari. Program yang hanya lima baris pembuktiannya bisa sehalaman, sehingga seringkali tidak pernah diterapkan dalam aplikasi nyata. Aktifitas ini dicakup pada matakuliah "Analisis Algoritma". Pada kuliah "Algoritma dan Pemrograman" mahasiswa sudah dipersiapkan untuk membuat spesifikasi "semi formal", sehingga hubungan dengan mata kuliah selanjutnya sudah dipersiapkan.

Aktifitas-aktifitas tersebut sebenarnya mencerminkan peran-peran seseorang dalam siklus hidup sebuah perangkat lunak.

Tujuan Kuliah Pemrograman Prosedural

Tujuan utama dari matakuliah ini adalah membekali mahasiswa cara berpikir dan pemecahan persoalan dalam paradigma pemrograman prosedural, serta membekali mahasiswa dengan modul dasar dari algoritma yang sering dipakai dalam pemrograman. Mahasiswa harus mampu membuat penyelesaian masalah pemrograman tanpa tergantung pada bahasa pemrograman apapun, dan kemudian ia mampu untuk mengeksekusi programnya dengan salah satu bahasa pemrograman prosedural yang sederhana. 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 prosedural dan menuliskan spesifikasi dan algoritmanya tanpa tergantung pada bahasa pemrograman apapun.
2. Menulis algoritma dari suatu masalah dengan menggunakan metodologi dan skema standard yang diajarkan secara terstruktur.

3. Menulis program yang "baik" (sesuai dengan kriteria yang diajarkan di kelas) dalam bahasa pemrograman yang diajarkan, dengan menggunakan aturan translasi yang diberikan. Programnya harus terstruktur walaupun bahasa pemrogramannya bukan bahasa yang terstruktur.

Mahasiswa harus mencoba terlebih dahulu algoritma yang akan dijelaskan di kelas, algoritma yang akan di bahas diberikan judulnya pada akhir kuliah sebelumnya. Setiap algoritma sebaiknya dicoba untuk diimplementasikan dengan bahasa tingkat tinggi yang dipelajari di kelas, dan mahasiswa pada akhir kuliah mempunyai pustaka yang lengkap, yang akan berguna dalam membuat tugas-tugas pemrograman pada kuliah lebih lanjut. Praktikum dengan komputer hanya akan dibimbing pada awal kuliah, dan untuk selanjutnya mahasiswa harus aktif melakukannya sendiri di laboratorium yang dibuka sepanjang hari kerja, atau di rumah jika mempunyai komputer pribadi.

Kuliah akan dipresentasi dengan kapur tulis biasa karena sebagian besar adalah proses kontruksi algoritma dan cara berpikir, yang dapat ditunjukkan dengan baik melalui papan tulis. Catatan kuliah yang telah dipersiapkan (diktat, *hands out*) dibuat agar banyak waktu terbuang untuk menyalin algoritma dari papan, karena itu harus dibaca.

Rujukan

Diktat ini dan diktat program kecil yang ditulis khusus untuk melengkapi diktat konsep, merupakan buku pegangan wajib dalam perkuliahan algoritma dan pemrograman di Departemen Teknik Informatika ITB.

Buku-buku teks (diurutkan sesuai abjad):

1. Aho, Hopcroft, Ullman: "Data Structures and Algorithms", Prentice Hall, 1987.
2. Horowitz, E. dan Sahni, S.: "Fundamentals of Data Structures in Pascal", Pitman Publishing Limited, 1984.
3. [Knuth-68] Knuth, D.E.: "The Art of Computer Programming", Vol. 1 : "Fundamentals Algoritms", Addison Wisley, 1968.
4. [Knuth-71] Knuth, D.E.: "The Art of Computer Programming", Vol. 3 : "Sorting and Searching", Addison Wisley, 1971
5. Meyer dan Baudoin: "Methodes de Programmation", Eyrolles, 1980.
6. [Scholl-88] Scholl P.C. dan Peyrin, J.P.: "Schemas Algorithmiques Fondamentaux", Masson, 1988.
7. Sedgewick R.: "Algorithms", Addison Wisley, 1984.
8. Wirth, N.: "Systematic programming", Prentice Hall, 1975.
9. [Wirth-86] Wirth, N.: "Algorithms & Data Stuctures", Prentice Hall, 1986.

Beberapa artikel yang perlu dibaca:

1. Djikstra, E.: "On the Cruelty of Really Teaching Computer Science", The SIGCSE Award Lecturer, SIGCSE Bulletin, vol.21, no.1, Feb 1989, pp xxiv – xxxix.
2. [Hoare-84] Hoare, C.A.R.: "Programming: Sorcery or Science?", IEEE Software, April 1984, pp 6 – 16.
3. Knuth, D. E.: "Programming As an Art", Communication of the ACM, Vol. 17, No. 12, Dec. 74, pp 667 – 673.

PENGERTIAN DASAR
Dalam Pemrograman Prosedural

Pada bagian ini akan dijelaskan definisi beberapa pengertian dasar yang penting sehubungan dengan algoritma dan pemrograman, yang akan diberikan dalam contoh pada kehidupan sehari-hari. Mungkin pengertian-pengertian tersebut mula-mula terasa abstrak bagi beberapa pembaca, tapi baiklah coba dipahami.

Pengertian pertama yang akan dijelaskan adalah **aksi**. Suatu aksi adalah kejadian yang terjadi pada suatu **selang waktu terbatas** dan menghasilkan **efek neto** yang telah terdefinisi dengan baik dan memang **direncanakan**. Pada deskripsi tersebut, ditekankan benar efek tersebut harus "direncanakan", maka berarti dititikberatkan pada kegunaannya. Jika seseorang tertarik pada suatu aksi, maka jelas bahwa minatnya adalah pada efek netonya.

Suatu aksi harus terjadi pada selang waktu yang terbatas, dimulai pada saat T_0 dan berakhir pada saat T_1. Maka efek neto dari aksi dapat dijelaskan dengan membandingkan keadaan pada saat T_0 dan keadaan pada saat T_1.

Contoh dari suatu aksi adalah Ibu Tati yang MENGUPAS KENTANG untuk mempersiapkan makan malam. Pernyataan ini mencakup hal yang luas ruang lingkupnya, misalnya:
- Apakah kentangnya harus dibeli dulu atau sudah ada di dapur?
- Apakah yang dimaksud dengan mengupas kentang untuk makan malam berarti sampai dengan kentang terhidang?
- Ketika kentangnya terhidang, jadi sup, digoreng, atau direbus saja?

Maka kita harus membatasi dengan jelas keadaan awal yang menjadi titik tolak mengupas kentang dan keadaan akhir yang ingin dicapai supaya dapat "merencanakan" efek neto yang diinginkan. Untuk itu ditentukan:
- *Initial state* (I.S.)/keadaan awal: T_0, adalah kentang sudah ada di kantong kentang, yang ditaruh di rak di dapur, di mana Ibu Tati akan mengupasnya
- *Final state* (F.S.)/keadaan akhir: T_1 kentang dalan keadaan terkupas di panci, siap untuk dimasak dan kantong kentangnya harus dikembalikan ke rak lagi.

Pengandaian yang lain adalah bahwa persediaan kentang si ibu selalu cukup untuk makan malam. Penambahan kentang ke dapur di luar tinjauan masalah ini. Ini adalah contoh bagaimana kita menentukan **batasan** dari persoalan yang akan diprogram.

Suatu kejadian dapat dipandang sebagai urut-urutan dari beberapa kejadian, berarti dapat diuraikan dalan beberapa (sub) aksi yang terjadi secara sekuensial. Dengan sudut pandang ini, maka efek kumulatifnya sama dengan efek neto dari seluruh kejadian. Dikatakan bahwa kejadian tersebut dianggap sebagai *sequential process* atau disingkat proses.

Penggolongan suatu kejadian sebagai proses atau aksi adalah relatif. Kejadian yang sama dapat dianggap sebagai aksi ataupun sebagai proses. Kalau lebih dititikberatkan pada efek netonya, yaitu keadaan " sebelum dan sesudah", maka kejadian tersebut

dianggap sebagai proses. Dengan menganggap kejadian tersebut suatu proses, T_0 adalah awal dari sebuah sub-aksi dan setiap akhir dari suatu sub-aksi akan merupakan awal dari sub-aksi berikutnya, dengan suatu keistimewaan, akhir dari sub-aksi yang terakhir adalah T_1 yaitu akhir dari seluruh kejadian.

Penggolongan suatu kejadian menjadi proses atau aksi tidak ada hubungannya dengan sifat dari kejadian itu sendiri melainkan tergantung dari cara peninjauan. Jika cara peninjauan dilakukan dari sudut pandang yang berbeda, maka biasanya hasil antara yang ingin diperhatikan juga berbeda.

Misalkan kejadian tentang Ibu Tati mengupas kentang, dapat dijelaskan oleh urut-urutan sub-aksi yang dilakukan oleh ibu tersebut, yaitu:

- *Ambil kantong kentang dari rak*
- *Ambil panci dari almari*
- *Kupas kentang*
- *Kembalikan kantong kentang dari rak*

Pada contoh tersebut, kejadian dijelaskan sebagai urut-urutan dari empat sub-aksi yang diungkapkan berdasarkan suatu pengamatan. Jika dari hasil pengamatan **tidak dipandang perlu** untuk menjelaskan bahwa kantong kentang diambil dari rak sebelum panci diambil dari almari, maka cukup dituliskan:

- *Ambil kantong kentang dari rak dan panci dari almari*
- *Kupas kentang*
- *Kembalikan kantong kentang ke rak*

Ada kejadian yang mempunyai suatu **pola tingkah laku**, atau disingkat **pola**. Kejadian tersebut akan terjadi jika pola ini diikuti. Efek neto dari kejadian ditentukan sepenuhnya oleh pola tersebut dan (mungkin) oleh keadaan awal (yaitu keadaan pada saat T_0). Kejadian yang lain mungkin mengikuti pola yang sama. Jika dua kejadian dengan pola yang sama menghasilkan efek neto yang berbeda, maka efek neto tersebut pasti tergantung pada keadaan awal, dan dapat dipastikan pula bahhwa keadaan awal dari keduanya berbeda.

Bagaimana cara mengamati pola yang sama dari berbagai kejadian, tidak dapat dijelaskan disini. Jika kita berjumpa dengan seorang teman, kita pasti segera dapat mengenalinya, apapun ekspresi yang sedang ditampilkannya. Mungkin ia sedang gembira, tertawa, menangis, atau bahkan ekspresi lain yang belum pernah ditampilkannya. Kita dapat mengenali teman tersebut karena kita kenal akan polanya. Demikian juga dengan jejadian yang berbeda, dapat pula dikenal pola-pola yang sama, walaupun disarikan dari keadaan awal dan efek neto yang mungkin berbeda. Mengenali pola ini sama halnya nanti dengan mengenali pola-pola solusi algoritmik untuk kelas persoalan tertentu yang akan dipelajari, menjadi bagian dari belajar memrogam.

Kembali ke contoh Ibu Tati yang mengupas kentang. pada suatu hari Ibu Tati mengupas kentang untuk malam dan kejadian tersebut kita amati. Keesokan harinya, ia mengupas kentang lagi untuk makan malam juga. Pada pengamatan yang kedua, kita **amati hal-hal yang sama** dengan hari sebelumnya. Dapatlah kita katakan: "Jelas, pengamatan tentang kedua kejadian akan **sama** karena ibu itu **mengerjakan hal-hal sama**?"

Pernyataan terakhir tersebut dapat benar atau salah, tergantung pada apa yang dimaksud dengan "mengerjakan hal yang sama". Untuk menyatakan "hal yang sama" harus hati-hati. Tinjaulah murid-murid sekolah dasar yang berpakaian sama, karena mereka memakai seragam. Apa yang ingin dinyatakan sebagai "sama" adalah bahwa baju dari setiap murid terbuat dari bahan yang sama dan modelnya sama pula, tanpa memperhitungkan kemungkinan adanya perbedaan ukuran tergantung dari perawakan setiap murid. Demikian pula, seorang murid dapat mempunyai lebih dari satu stel seragam yang sama.

Kedua dari aksi Ibu Tati mengupas kentang pada dua hari yang berlainan tersebut juga dapat dipandang berbeda, seperti halnya baju murid sekolah dasar tersebut. Kejadian yang satu terjadi pada hari Sabtu dan yang lain pada hari Minggu. Karena setiap kentang hanya dapat dikupas satu kali, maka kentang yang terlibat pada kedua kejadian tersebut "berbeda" pula. Pada hari Minggu, mungkin kantong kentangnya berisi lebih sedikit dari kemarinnya, dan sebagainya.

Tetapi kedua kejadian mengupas kentang pada hari Sabtu dan Minggu dapat pula dikatakan sama karena kemiripannya, dan disepakati untuk memberi nama yang sama untuk kedua aksi tersebut, misalnya "MENGUPAS KENTANG UNTUK MAKAN MALAM".

Algoritma adalah deskripsi dapat terdiri dari suatu pola tingkah laku, dinyatakan dalam **primitif**, yaitu aksi-aksi yang didefinisikan sebelumnya dan diberi nama, dan diasumsikan sebelumnya bahwa aksi-aksi tersebut dapat dikerjakan sehingga dapat menyebabkan kejadian yang dapat diamati.

Suatu algoritma dapat terdiri dari beberapa sub-algoritma, jika setiap sub-aksi juga dapat diuraikan dalam urut-urutan yang dapat dimengerti dengan baik dan terbatas.

Pengertian algoritma, yaitu sebagai suatu petunjuk untuk mewujudkan suatu efek neto, telah sangat dikenal dalam kehidupan sehari-hari. Petunjuk untuk merajut, resep-resep kue, aturan pakai suatu peralatan elektronik, adalah algoritma, yaitu deskripsi dari pola tingkah laku yang jika dikerjakan akan membawa ke tujuannya.

Aksi primitif **harus dapat dikerjakan**. "Pergi ke seberang jalan!" adalah aksi yang dapat dikerjakan, sedangkan "Pergi ke Neraka!" bukan algoritma karena tidak dapat dikerjakan.

Urut-urutan langkah harus dapat dimengerti dengan baik, oleh pembuat algoritma maupun oleh yang akan mengerjakan. Tidak boleh ada sedikit pun salah pengertian di antara keduanya supaya dapat dihasilkan efek yang diinginkan.

Jika pada suatu resep kue dituliskan "Panaskan dulu oven", maka instruksi tersebut tidak jelas karena berapa lama dan sampai temperatur oven mencapai berapa derajat hal tersebut harus dilakukan, tidak ditentukan dengan pasti.

Sekarang perhatikanlah laporan pengamatan tentang kejadian Ibu Tatiyang mengupas kentang:

- *Ibu Tati mengambil kantong kentang dari rak*
- *Ibu Tati mengambil panci dari almari*
- *Ibu Tati mengupas kentang*
- *Ibu Tati mengembalikan kantong kentang ke rak*

Bandingkanlah hasil pengamatan di atas dengan teks berikut, yang merupakan algoritma, yaitu sekumpulan intsruksi yang diberikan oleh Ibu Tati kepada pembantu barunya:

- *Ambil kantong kentang dari rak*
- *Ambil panci dari almari*
- *Kupas kentang*
- *Kembalikan kantong kentang ke rak*

Jika teks algoritma tersebut diberikan kepada pembantunya yang bernama Ina, maka jika dilaksanakan akan menghasilkan pengamatan kejadian :

- *Ina mengambil kantong kentang dari rak*
- *Ina mengambil panci dari almari*
- *Ina mengupas kentang*
- *Ina mengembalikan kantong kentang ke rak*

Atau jika putri sulung Ibu Tati yang bernama Aida pada suatu hari dengan senang hati mengerjakan pengupasan kentang, maka akan dihasilkan pengamatan kejadian :

- *Aida mengambil kantong kentang dari rak*
- *Aida mengambil panci dari almari*
- *Aida mengupas kentang*
- *Aida mengembalikan kantong kentang ke rak*

Dengan membandingkan teks hasil pengamatan terhadap algoritma, dapat ditarik kesimpulan: algoritma Ibu Tati menyatakan cara-cara untuk melakukan sesuatu sedangkan laporan pengamatan menjelaskan tentang kejadian itu sendiri. Adakah kesimpulan yang lain? Tentu saja tidak ada, jika kita batasi bahwa algoritma yang diberikan adalah sederetan aksi-aksi bernama, yang harus dikerjakan dengan urutan tertentu. Dengan batasan ini, pengamat-pengamat dapat melaporkan dengan baik suatu aksi sesuai yang terjadi. Tetapi kelakuan Ibu Tati (atau pembantu) dapat lebih rumit. Misalnya sehabis mengambil panci ia memakai celemek **jika perlu**, yaitu jika kebetulan ia memakai baju berwarna muda. Maka pada suatu hari ia memakai celemek, sedangkan pada hari lain tidak.

Secara umum dapat menyebut tentang celemek dan kondisi yang menyebabkan celemek tersebut dipakai, satu laporan pengamatan dapat ditulis unutk setiap kejadian: Misalnya suatu hari, dihasilkan laporan pengamatan sebagai berikut :

- *Ibu Tati mengambil kantong kentang dari rak*
- *Ibu Tati mengambil panci dari almari*
- ***Ibu Tati memakai celemek***
- *Ibu Tati mengupas kentang*
- *Ibu Tati mengembalikan kantong kentang ke rak*

atau pada suatu hari yang lain, dihasilkan laporan pengamatan yang tidak sama dengan sebelumnya:

- *Ibu Tati mengambil kantong kentang dari rak*
- *Ibu Tati mengambil panci dari almari*
- *Ibu Tati mengupas kentang*
- *Ibu Tati mengembalikan kantong kentang dari rak*

Sekarang, masalahnya adalah bagaimana menuliskan teks pengamatan yang sama dari kedua laporan pengamatan yang berbeda tersebut, misalnya:

- *Ambil kantong kentang dari rak*
- *Ambil panci dari almari*
- ***Lakukan persiapan, tergantung pakaian***
- *Kupas kentang*
- *Kembalikan keranjang kentang ke rak*

Dengan pengertian implisit "lakukan persiapan tergantung pakaian" menyertakan tidak ada aksi jika pakaian tidak berwarna muda dan menyatakan pemakaian celemek jika pakaian berwarna muda.

Tetapi jika diinginkan lebih terinci dan ingin menyebut secara eksplisit maka **lakukan persiapan tergantung pakaian** harus diganti dengan hasil pengamatan pada hari yang bersangkutan.

Maka pada hari Sabtu :

- *Ibu Tati melihat bahwa bajunya tidak berwarna muda karena itu ia **tidak memakai celemek** (berarti tidak ada aksi memakai celemek)*

Sedangkan laporan pada hari Minggu:

- *Ibu Tati melihat bahwa bajunya berwarna muda karena itu ia **memakai celemek***

Pada derajat yang rinci tidak mungkin kedua kejadian ini dilaporkan dalam satu laporan pengamatan, karena terperinci, kedua kejadian tersebut berbeda.

Inilah algoritma, yaitu menyatakan pola tingkah laku yang sama untuk dua, bahkan tak berhingga kejadian yang berbeda dan dengan menjelaskan pola tersebut memberikan sesuatu yang daoat terjadi pada suasana lingkungan apapun (dalam contoh tersebut, baju warna gelap ataupun muda). Apa yang sebenarnya terjadi jika suatu pola tingkah laku diikuti dapat ditentukan pola oleh keadaan yang berlaku ketika aksi tersebut mulai.

Ada dua hal yang penting. Pertama, **pengamatan apakah** baju si ibu berwarna muda, dan kedua berdasarkan pengamatan tersebut **aksi** "memakai celemek" bisa terjadi atau tidak (berarti aksi tersebut kondisional). Maka notasi untuk aksi kondisional dinyatakan oleh **kondisi** dan **aksi**.

- *Ambil kantong kentang dari rak*
- *Ambil panci dari almari*
- **if** *baju berwarna* muda **then**
  Pakai celemek
- *Kupas kentang*
- *Kembalikan kantong ke rak*

Maka aksi kondisional mengandung dua aksi, aksi pertama harus suatu **pengamatan**. Hasil dari pengamatan ini adalah suatu keadaan benar ("*true*") atau salah ("*false*"). Aksi kedua menghasilkan kejadian berlangsung sesuai dengan hasil pengamatan. Jika pengamatan memberikan hasil "*true*", maka aksi terjadi, jika pengamatan memberikan hasil "*false*", maka aksi tidak dilakukan.

Selain notasi untuk aksi kondisional, kita perlukan lagi beberapa notasi yang menunjukkan bahwa algoritma lebih tinggi tingkatannya dan menyangkut abstraksi dari pengamatan, yaitu notasi yang mewakili proses **pengulangan**. Misalnya kita ingin menyatakan bahwa "mengupas kentang" adalah suatu proses mengerjakan **satu** buah kentang pada suatu saat, maka aksi primitif kita adalah "kupas 1 kentang". Jika jumlah kentang yang ingin dikupas selalu sama setiap hari, misalnya 25 maka sebagai ganti "kupas kentang" dapat dituliskan 25 kali "kupas 1 kentang", masing-masing pernyataan dituliskan per baris sehingga keseluruhannya dituliskan dalam 25 baris sekuensial. Tetapi jika kentang yang dikupas tidak selalu sama (dan hal ini lebih sering terjadi dalam kenyataan) sedangkan kita tetap menginginkan pola kelakuan yang sama apa yang harus dilakukan? Setiap kali kita harus mengganti teks, satu jenis teks untuk satu kali pengamatan. Ini bukan tujuan dari penulisan algoritma yang mampu menghasilkan pengamatan yang berbeda-beda. Dianggap bahwa si ibu mampu untuk melongok ke panci dan dengan demikian mengamati apakah kentang yang dibutuhkan telah cukup.

Jika diketahui bahwa kasus yang ekstrem adalah mengupas 500 kentang (karena kentangnya sangat kecil-kecil dan ada pesta), artinya Ibu Tati tidak mungkin mengupas lebih dari 500 kentang, kita dapat menuliskan algoritma umum untuk mengupas kentang dengan menuliskan 500 (lima ratus) kali secara sekuensial pernyataan berikut:

- **if** *jumlah kentang yang sudah dikupas (belum cukup)* ***then***
  Kupas 1 kentang

Siapa pun pasti merasa keberatan dengan cara penulisan semacam itu, yaitu harus menuliskan hal yang sama 500 kali. Dengan asumsi dasar bahwa sebelumnya harus diketahui berapa jumlah kentang yang harus dikupas, batas seperti itu terlalu besar untuk rata-rata yang terjadi. Jika sebenarnya hanya diinginkan mengupas 25 buah kentang, maka pengamatan ke-26 akan memberikan hasil "*false*" yang pertama, dan 474 pengamatan berikutnya tidak akan memberikan hasil pengamatan yang baru. Sekali si ibu telah tahu bahwa kentang yang dikupasnya cukup, tidak perlu lagi memaksa dia melongok ke panci 474 lagi untuk meyakinkan dirinya sendiri.

Untuk mengatasi hal ini, diperkenalkan suatu notasi yang menjelaskan tentang suatu proses **pengulangan** sampai dijumpai keadaan tertentu, dan dituliskan sebagai :

while *(kondisi)* ***do***
Aksi

Dengan notasi ini algoritma mengupas kentang dapat dituliskan :

- *Ambil kantong kentang dari rak*
- *Ambil panci dari almari*
- **if** *baju berwarna muda then*
  pakai celemek
- **while** *jumlah kentang terkupas belum cukup* **do**
  Kupas 1 kentang
- *Kembalikan kantong kentang ke rak*

Contoh berikut, akan dijelaskan pola kelakuan dari Ibu Tati yang menggunakan primitif sama, yang karena alasan tertentu selalu mengupas kentang dengan **jumlah genap** untuk masakannya. Maka dapat dituliskan algoritma sebagai berikut:

- *Ambil kantong kentang dari rak*
- *Ambil panci dari almari*
- **if** *baju berwarna muda* **then**
  pakai celemek
- **while** *jumlah kentang terkupas belum cukup* **do**
  Kupas 1 kentang
  Kupas 1 kentang
- *Kembalikan kantong kentang ke rak*

Contoh di atas menunjukkan bahwa aksi primitif yang sama dapat menggambarkan pola kelakuan yang berbeda.

Berikut ini diandaikan bahwa Ibu Tati dalah penggemar kentang sehingga ia selalu mempunyai beberapa kantong kentang di raknya. Kantong kentangnya kemungkinan ada yang berisi ataupun kosong.

Jika pengupasan kentang dapat dilakukan dari beberapa kantong, dapat dituliskan algoritma untuk mengupas sejumlah tertentu kentang sebagai berikut:

- *Ambil kantong kentang dari rak*
- *Ambil panci dari almari*
- **depend on** *baju*
  berwarna muda : pakai celemek
  tidak berwarna muda : -
- **while** *jumlah kentang terkupas belum cukup* **do**
  depend on *kantong kentang*
  ada isinya : Kupas 1 kentang
  tidak ada isinya : Ambil kantong kentang lain dari rak
  Kupas 1 kentang

Dari contoh yang terakhir dapat pula ditarik kesimpulan bahwa suatu algoritma dapat dibangun dari aksi primitif dan gabungan dari notasi standard yang telah kita kenal. Satu algoritma mewakili beberapa kejadian yang berdasarkan pengamatan berbeda.

Algoritma adalah suatu sebuah teks yang tidak tergantung waktu, konsepnya statik. Di pihak lain ada realisasi kejadian yang dicakup oleh algoritma tersebut, yaitu suatu eksekusi yang dinamik, terjadi tergantung pada waktu, yang dijelaskan sebagai hasil dari pengamatan.

Telah dikatakan bahwa aksi harus terjadi dalam selang waktu yang terbatas, maka algoritma yang menjelaskan tentang aksi tersebut harus mencakup hal tersebut.

Kita tidak boleh menuliskan:

- **while** *pakaian berwarna muda* **do**
  Kupas 1 kentang berikutnya

Karena pengupasan kentang tidak mempengaruhi warna pakaian, hanya ada duia kemungkinan: pakaian tidak berwarna muda dan pengupasan kentang **tidak pernah dilakukan** atau pakaian berwarna muda dan **proses pengupasan kentang akan dilakukan terus menerus**, yang berarti bahwa jika kebetulan pakaian berwarna muda, maka pengamatan akan menghasilkan sesuatu yang tidak pernah berhenti

(*looping*). Contoh ini adalah sebuah algoritma yang salah karena dapat mengakibatkan pengulangan yang tidak pernah berhenti.

Tidak mudah untuk menentukan apakah suatu teks yang tampak seperti sebuah algoritma adalah memang algoritma yang benar. Bahkan **tidak mungkin** untuk membuat sebuah algoritma yang mamapu memeriksa suatu teks dan menentukan apakah teks tersebut suatu algoritma yang benar. Maka adalah tanggung jawab moral orang-orang yang profesinya membuat algoritma untuk mempersiapkan bukti bahwa algoritma yang dibuatnya adalah sebuat algoritma yang benar.

Pengertian dasar yang lain adalah tentang **mesin.** Suatu mesin adalah sebuah **mekanisme** yang dapat menyebabkan suatu aksi terjadi mengikuti suatu pola tingkah laku yang dijelaskan oleh algoritma yang urut-urutan pelaksanaannya dinyatakan dalan aksi primitif mesin tersebut.

Pada contoh-contoh di atas, diberikan beberapa algoritma tentang mengupas kentang. Semua algoritma dinyatakan dalam primitif yang sama yang kemudian melahirkan teks yang dinyatakan sebagai "pengamatan kejadian". Siapapun yang mampu untuk:
- Mengerjakan aksi primitif tersebut,
- Menerima algoritma yang dinyatakan dengan primitif tersebut dan akan melaksanakan langkah-langkah dengan patuh,disebut sebagai **mesin.**

Jika saya dapat membuat teman saya, pembantu saya, tetangga kiri saya, atau tetangga kanan saya mengerjakan pengupasan kentang tersebut tergantung algoritma yang saya berikan, maka teman, pembantu, tetangga kiri maupun tetangga kanan saya adalah sebuah mesin.

Suatu mekanisme yang hanya dapat mengerjakan satu hal yang selalu sama (misalnya *toilet flusher*) tidak dapat disebut suatu mesin. Hal penting yang dapat dikerjakan oleh mesin adalah aksi-aksi yang sekuensial, kemampuan menerima suatu pola tingkah laku dan berkelakuan berdasarkan pola tersebut. Mesin adalah **pengeksekusi atau pelaku dari algoritma**.

Algoritma yang mengontrol pola tingkah laku suatu mesin disebut program. Dengan perkataan lain, program adalah algoritma yang dimaksudkan untuk dieksekusi oleh mesin. Dalam pengertian algoritma yang harus dapat dimengerti dengan baik, tanpa menghiraukan bagaimana pengertian tersebut diwujudkan. Mesin terdiri dari sekumpulan peralatan. Berkat konstruksinya, mesin hanya dapat mengerjakan sekumpulan instruksi-instruksi tertentu yang terbatas jumlahnya yang telah terdefinisi. Jika kita berikan suatu program kepada mesin, maka dengan patuh ia akan mengerjakan persis seperti yang dinyatakan dalam algoritma. Mesin sangat "patuh" terhadap instruksi yang kita berikan, dengan resiko kita harus mendefinisikan instruksi tersebut dengan rinci. Untuk seorang pemrogram yang belum berpengalaman, hal ini sering menimbulkan keluhan, tetapi dengan bertambahnya pengalaman, ia akan menyadari bahwa mesin akan selalu mengerjakan hal-hal yang dianggap "tidak umum" tanpa membantah. Bandingkanlah dengan pembantu yang seringkali mengadakan penafsiran sendiri terhadap instruksi kita supaya ia lebih enak, tetapi sering malahan menjengkelkan dan menyulitkan kita karena interpretasi dan tingkah lakunya yang tak terkontrol oleh kita.

Program yang mengontrol mesin harus disusun sedemikian rupa jika ingin dipakai sesuai keinginan. Misalnya, kita ingin menyuruh mesin tersebut untuk memecahkan masalah yang kita hadapi, maka kita jarus membuat program yang sesuai untuk masalah tersebut dan mesin akan mengerjakan program sesuai dengan algoritma yang ditulis. Dalam hal ini mesin tersebut adalah alat bantu untuk memecahkan dari sudut pandang ini.

Pada bagian berikutnya akan dijelaskan, apakah pemrograman itu, dan mesin pengeksekusi program yang selanjutnya disebut sebagai komputer. Primitif-primitif yang akan diuraikan pada bab-bab selanjutnya adalah primitif yang dapat dilakukan oleh komputer. Aksi primitif komputer terlalu detil untuk jalan pikiran manusia, sehingga terlalu sulit untuk menguraikan algoritma dalam primitif langsung komputer. Karena itu diperlukan mesin abstrak, yaitu mekanisme yang diasumsikan dapat dilakukan oleh sekumpulan primitif yang diberikan. Berangsur-angsur, secara bertahap, mesin abstrak akan dijabarkan menjadi mesin riil, yaitu sampai primitif yang dapat dilakukan oleh komputer. Keadaan awal dan keadaan akhir yang diceritakan di atas pada kejadian nyata, juga akan diwakili oleh keadaan komputer, dalam hal ini keadaan isi memori. Efek neto yang akan dihasilkan dinyatakan dalam spesifikasi program, yang menjadi bahan mentah dalam menuliskan programnya. Dari kejadian sehari-hari yang diuraikan pada bab ini, kita akan berbicara dalam notasi algoritmik. Notasi kondisional dan pengulangan di atas baru sebagian dari notasi algoritmik yang akan dipelajari secara bertahap (karena itu dituliskan dalam Bahasa Inggris), untuk membedakan dengan kalimat-kalimat Ibu Tati dalam bahasa manusia.

Latihan Soal

I. Periksalah apakah masing-masing algoritma "mengupas kentang" berikut benar:
Pada algoritma ini kita hanya tertarik pada aksi mengambil kentang dan kantong dan mengupas kentang.

1. *while jumlah kentang terkupas belum cukup do*
   depend on kantong
   kantong berisi 1 kentang : Kupas 1 kentang

2. *while kantong tidak kosong do*
   Kupas 1 kentang
   Kupas 1 kentang

3. *while kantong tidak kosong do*
   while jumlah kentang terkupas belum cukup do
   Kupas 1 kentang

4. *while (kantong tidak kosong) dan (jumlah kentang terkupas belum cukup) do*
   Kupas 1 kentang

5. *while jumlah kentang terkupas belum cukup do*
   depend on kantong
   kantong tidak kosong : kupas 1 kentang
   kantong kosong : ambil kantong lain

6. *while jumlah kentang terkupas belum cukup do*
   while kantong tidak kosong do
   Kupas 1 kentang

7. *while jumlah kentang terkupas belum cukup do*
   depend on kantong
   kantong tidak kosong : Kupas 1 kentang
   kantong kosong : -

8. *while kantong ada isinya do*
   depend on jumlah kentang terkupas
   jumlah kentang terkupas belum cukup : Kupas 1 kentang
   jumlah kentang terkupas cukup : Ambil kantong lain
   Kupas 1 kentang

9. *while kantong ada isinya do*
   depend on jumlah kentang terkupas
   jumlah kentang terkupas belum cukup : Kupas 1 kentang
   jumlah kentang terkupas cukup : Stop

10. *depend on kantong*
    kantong tidak kosong : while jumlah kentang terkupas belum cukup do
    kupas 1 kentang
    kantong kosong : beli kentang lagi

11. Pada algoritma ini berlaku hipotesis: mula-mula kantong penuh
    Kupas 1 kentang
    depend on jumlah kentang terkupas
    jumlah kentang terkupas belum cukup : kupas 1 kentang
    jumlah kentang terkupas cukup : kembalikan kantong ke raknya

12. *while kantong ada isinya do*
    Kupas 1 kentang
    depend on jumlah kentang terkupas
    jumlah kentang terkupas belum cukup : cari kentang lagi
    jumlah kentang terkupas cukup : -

13. *while (jumlah kentang terkupas belum cukup) dan (kantong ada isinya) do*
    kupas 1 kentang

14. *depend on kantong*
    kantong ada isinya :
    while jumlah kentang terkupas belum cukup do
    Kupas 1 kentang
    kantong tidak ada isinya :
    Taruh kentang dalam kantong
    while jumlah kentang terkupas belum cukup do
    Kupas 1 kentang

II. Distribusi gula-gula

a. Ada sekantong gula-gula, hendak dibagikan merata ke empat orang anak. Tiap anak harus mendapat jumlah yang sama, dan jika sisanya tidak cukup untuk dibagikan ke empat anak tersebut, maka sisanya tidak dibagikan. Tuliskanlah algoritmanya.

b. Jika gula-gula tersebut mempunyai rasa jeruk, mentol, arbei dan durian, dan setiap anak harus mendapat jumlah dan rasa yang sama, tuliskan pula algoritma untuk membaginya.

   Catatan: Definisikan dahulu aksi primitif untuk persoalan ini.

NOTASI ALGORITMIK

Dalam perkuliahan ini, akan dipakai sebuah notasi yang akan dipakai sebagai *standard* dalam menuliskan teks algoritma. Dalam kuliah ini dibedakan antara algoritma dan program. Algoritma adalah solusi detail secara prosedural dari suatu persoalan dalam notasi algoritmik. Program adalah program komputer dalam suatu bahasa pemrograman yang tersedia di dunia nyata. Bahasa komputer mempunyai pemroses sehingga dapat dieksekusi mesin, sehingga teks program dibuat untuk dieksekusi mesin (dan untuk kepentingan pemeliharaan program sebaiknya dapat dibaca dengan mudah oleh manusia). Notasi algoritmik yang dipakai di kuliah ini diadaptasi dari [Scholl-88], merupakan notasi yang sengaja dikembangkan untuk kepentingan pengajaran di lingkungan Program Studi Teknik Informatika ITB, dan tidak mempunyai mesin pengeksekusi. Notasi ini dianggap perlu untuk menjembatani keragaman dan kompleksitas bahasa sehingga mahasiswa mampu melakukan "abstraksi". Notasi ini akan merangkum semua konsep pemrograman prosedural yang harus dapat dengan mudah dituliskan di atas kertas. Notasi ini lebih berorientasi kepada *detail design* dibandingkan *coding*. Notasi ini hanyalah alat untuk menuangkan rancangan secara prosedural yang selanjutnya dengan mudah dapat ditranslasi menjadi salah satu program dalam bahasa tertentu. Suatu saat, jika permogram menghadapi dunia profesional yang membutuhkan hasil yang siap pakai, notasi dapat disesuaikan dan dibuat lebih dekat dengan bahasa pemrograman yang dipakai.

Teks algoritma selalu terdiri dari tiga bagian, yaitu :
- Judul (Header)
- Kamus
- Algoritma

Pada setiap bagian tersebut, akan didefinisikan dan dipakai nama, atau dituliskan komentar dalam bahasa Indonesia. Komentar dalam bahasa Indonesia dituliskan di antara tanda kurung kurawal. Teks yang tidak dituliskan di antara kurung kurawal buka dan tutup adalah teks dalam notasi algoritmik.

Contoh teks algoritmik:

```
JUDUL
{ Ini adalah teks dalam bahasa Indonesia untuk memudahkan pembacaan
teks algoritma }
{ Spesifikasi teks algoritmik secara umum }
```

```
KAMUS
{ Pada bagian ini, dilakukan pendefinisian nama konstanta, nama
variabel, spesifikasi prosedur, spesifikasi fungsi }
```

```
ALGORITMA
{ Pada bagian ini, semua teks yang tidak dituliskan di antara tanda
kurung kurawal buka dan kurung kurawal tutup harus dianggap sebagai
notasi algoritmik }
```

Nama

Nama dalam sebuah teks algoritmik adalah **sesuatu** yang dipakai sebagai identifikasi:
- modul **program**, **algoritma**, **skema** program, dan sebagainya,
- **fungsi**,
- **prosedur,**
- **type,**
- **tempat penyimpanan**, supaya harga yang disimpan dapat diacu isinya. Dalam beberapa bahasa pemrograman, nama tempat penyimpanan ini seringkali disebut sebagai **nama variabel** karena isinya dapat diubah-ubah lewat instruksi program.
- **konstanta**, yaitu suatu harga yang tetap dan tidak boleh diubah nilainya.

Karena adanya bermacam-macam nama tersebut, dalam suatu teks algoritma dikenal nama program, nama skema, nama fungsi, nama prosedur, nama type, nama variabel dan nama konstanta. Semua nama dalam program harus unik, artinya suatu nama hanya didefinisikan satu kali (dipakai sebagai salah satu nama tersebut) satu kali saja. Namun sebuah nama, misalnya nama variabel boleh dipakai berkali-kali dalam beberapa instruksi.

Semua nama yang dipakai dalam suatu teks algoritma harus sudah didefinisikan pada salah satu bagian teks algoritma.

Tatacara penamaan yang dipakai dalam notasi algoritmik tidak seketat dalam bahasa pemrogaman yang misalnya membatasi jumlah dan jenis karakter serta ***case sensitive*** (huruf kecil dan kapital dibedakan). Namun sebaiknya nama yang dipakai tidak membingungkan, misalnya dalam teks algoritmik, nama tidak dibedakan dari penulisan dengan huruf kecil/kapital (tidak ***case sensitive***) atau memakai simbol operator sebagai bagian dari nama.

Judul (*Header*)

Judul adalah bagian teks algoritma tempat mendefinisikan apakah teks tersebut adalah program, prosedur, fungsi, modul atau sebuah skema program. Setelah judul disarankan untuk menuliskan spesifikasi singkat dari teks algoritma tersebut. Pada bagian judul dan spesifikasi, pembaca dapat mengetahui isi dari teks tanpa membaca secara detil Bagian judul berisi judul teks algoritmik secara keseluruhan dan intisari sebuah teks algoritmik tersebut. Bagian judul ini identik dengan judul buku dan intisari pada sebuah teks ilmiah dalam suatu makalah berbahasa Indonesia.

Kamus

Kamus adalah bagian teks algoritma tempat mendefinisikan:
- nama type,
- nama konstanta,
- nama informasi (nama variabel),
- nama fungsi, sekaligus spesifikasinya,
- nama prosedur, sekaligus spesifikasinya.

Semua **nama** tersebut baru dapat dipakai jika didefinisikan dalam **kamus**. Penulisan sekumpulan nama dalam kamus sebaiknya dikelompokkan menurut jenis nama tersebut.

Nama variabel belum terdefinisi harganya ketika didefinisikan. Pendefinisian nama konstanta sekaligus memberikan harganya. Pendefinisian nama fungsi dilakukan sekaligus dengan domain dan range serta spesifikasinya. Pendefinisian nama prosedur sekaligus dengan pendefinisian parameter (jika ada) dan spesifikasi prosedur *(initial state*, *final state*, dan proses yang dilakukan).

Dalam bahasa pemrograman, setiap **nama** mempunyai aturan penulisan (sintaks) tertentu, misalnya yang menyangkut karakter yang diperbolehkan, jumlah maksimum karrakter, dsb. Pada teks algoritma, tidak ada aturan ketat mengenai nama. Yang penting adalah bahwa pemilihan nama harus interpretatif, tidak menimbulkan kerancuan, dan jika singkat, harus disertai dengan penjelasannya. Karena nama merupakan satu kesatuan ***leksikal***, sebuah nama harus dituliskan secara utuh (tidak boleh dipisahkan dengan *blank*) supaya satu nama dapat dibedakan dari nama yang lain atau satuan leksikal lain. Nama informasi sebaiknya menunjukkan type. Contoh nama yang menimbulkan kerancuan: X-Y akan membingungkan sebab mungkin berarti X “minus” Y.

Kamus global atau umum dikenal untuk seluruh program. Kamus lokal hanya dikenal pada teks algoritma di mana kamus tersebut ditulis.

Contoh pendefinisian kamus:

```
KAMUS
{ Nama Type, hanya untuk type yang bukan type dasar }
   type  Point : <X : real, Y : real>     { koordinat pada sumbu
                                            kartesian }

{ Nama Konstanta, harus menyebutkan type dan nilai }
   constant PI :  real  = 3.14159
   constant Ka : character = 'K'
   constant Faktor : integer = 3
   constant Benar : boolean = true

{ Nama Informasi, menyebutkan type}
   NMax : integer   { jumlah maksimum elemen tabel }
   Found : boolean  { Hasil pencarian, true jika ketemu }
   P : Point        { Posisi pena pada bidang kartesian }
   CC : character   { Current character }

{ Spesifikasi fungsi, menyebutkan nama fungsi, domain, dan range }
   function RealToInt (i : real) → integer
   { Mengkonversi harga i yang bertype real menjadi harga ekivalen
     yang bertype integer }

{ Spesifikasi Prosedur, menyebutkan Nama, parameter, Initial State
  (I.S.), Final State (F.S.), dan Proses }
   procedure INISIALISASI
   { I.S. Sembarang
     F.S. Semua nama global dalam kamus terdefinisi nilainya
     Proses : Menginisialisasi semua nama informasi global }
   procedure Tulis (input Pesan : string)
   { I.S. Sembarang
     F.S. Pesan tertulis di layar
     Proses : Menulis isi Pesan ke layar }
   procedure TUKAR (input/output A, B : real)
   { I.S. A dan B terdefinisi, A=a dan B=b
     F.S. A=b dan B=a
     Proses : Menukar isi nama informasi A dan B }
```

Algoritma

Algoritma adalah bagian teks algoritmik yang berisi instruksi atau pemanggilan aksi yang telah didefinisikan. Komponen teks algoritmik dalam pemrograman prosedural dapat berupa:

- instruksi dasar seperti *input/output*, *assignmnet*,
- *sequential statement*,
- analisis kasus,
- pengulangan.

Contoh suatu teks algoritma yang lengkap dapat dilihat pada bab-bab berikutnya.

TYPE

Type adalah pola representasi suatu data dalam komputer. Gunanya untuk mendefinisikan objek yang akan diprogram. Ada type dasar (yang diasumsikan ada) dan type bentukan, biasanya type bentukan dibentuk dari type dasar. Type tidak menentukan alokasi memori di komputer, tetapi hanya mendefinisikan pola struktur informasi.

Mendefinisikan TYPE berarti :
- menentukan **nama type** dalam kamus,
- definisi **domain harga** yang dapat dipunyai oleh nama tersebut,
- konvensi atau perjanjian tentang penulisan **konstanta** bertype tersebut,
- **operator** yang dapat dioperasikan terhadap objek bertype tersebut.

Ada type dasar yang sudah diberikan dan dan siap dipakai, ada type bentukan yang dibentuk dari **type dasar** atau dari **type bentukan/komposisi** yang sudah dibuat.

Type Dasar

Type dasar yang tersedia dalam suatu bahasa adalah type yang sudah didefinisikan oleh pemroses bahasa. Karena sudah didefinisikan, pemrogram dapat memakai nama type dan semua operator yang tersedia, dan mentaati domain nilai yang disimpan dalam type tersebut.

Type dasar yang dianggap biasanya tersedia dalam suatu bahasa tingkat tinggi (dan merupakan type dasar dalam notasi algoritmik) adalah type-type dasar berikut:
- bilangan logika/boolean
- bilangan bulat
- bilangan riil
- karakter

Implementasi type tersebut dalam berbagai bahasa dapat sedikit berbeda. Akan dipelajari ketika dijelaskan pada bahasa yang bersangkutan.

Bilangan Logika/Boolean

Nama : boolean
Domain : [true, false]
Konstanta : true false
Operator :

KELOMPOK	Op	ARTI	HASIL
Operator logika	and	dan	boolean
	or	atau	boolean
	xor	ekslusive OR	boolean
	not	negasi	boolean
	EQ	ekivalensi	boolean
	nEQ	Negasi dari ekivalensi	boolean

Hasil operasi operator boolean tersebut diberikan oleh tabel kebenaran sebagai berikut:

Operasi	Hasil
true and true	true
true and false	false
false and true	false
false and false	false
not false	true
not true	false
true EQ true	true
true EQ false	false
false EQ true	false
false EQ false	true

Operasi	Hasil
true or true	true
true or false	true
false or true	true
false or false	false
true xor true	false
true xor false	true
false xor true	true
false xor false	false

Catatan: Dalam beberapa bahasa pemrograman, didefinisikan operator logika yang dihubungkan dengan eksekusi dan evaluasi ekspresi, seperti operator **and then** dan **or else** (lihat Diktat "Pemrograman Fungsional" untuk penjelasan operator ini). Pada diktat ini, operator and dan or merupakan operator murni logika seperti dituliskan pada tabel di atas.

Bilangan Bulat

Nama : **integer**
Domain : **Z** (hati-hati dengan representasi komputer)
Konstanta : 0 3 123 -89 56 999

Bilangan integer mempunyai keterurutan. Keterurutan ini didefinisikan dengan :
- suksesor x adalah x + 1
- predesesor x adalah x - 1

Contoh:

suksesor 0 adalah 1
suksesor -1 adalah 0
suksesor 5 adalah 6

predesesor -1 adalah -2
predesesor 3 adalah 2

Operator :

KELOMPOK	Op	ARTI	Hasil
Operator aritmatika	*	Kali	integer
	+	Tambah	integer
	-	Kurang	integer
	/	Bagi	real
	div	Bagi	integer
	mod	Sisa pembagian bulat	integer
	abs	Nilai absolut	integer >0
	^	Pangkat	integer

KELOMPOK	Op	ARTI	Hasil
Operator relasional/ perbandingan	<	Lebih kecil	boolean
	≤	Lebih kecil atau sama dengan	boolean
	>	Lebih besar	boolean
	≥	Lebih besar atau sama dengan	boolean
	=	Sama dengan	boolean
	≠	Tidak sama dengan	boolean

Bilangan Riil

Nama : **real**
Domain : **R** (hati-hati dengan representasi komputer)
Konstanta : **Angka mengandung '.'**
Dapat dituliskan dengan notasi E yang berarti pangkat sepuluh.
Contoh konstanta:
```
0.   0.2   3.233 123.  -89.0 56.   999.  12.E-2    1.5E1
```
Operator :

KELOMPOK	Op	ARTI	Hasil
Operator aritmatika	*	Kali	real
	+	Tambah	real
	−	Kurang	real
	/	Bagi	real
	^	Pangkat	real
Operator relasional/ perbandingan	<	Lebih kecil	boolean
	≤	Lebih kecil atau sama dengan	boolean
	>	Lebih besar	boolean
	≥	Lebih besar atau sama dengan	boolean
	≠	Tidak Sama dengan	boolean

Catatan:
1. Bilangan riil yang mengandung E berati pangkat sepuluh. Contoh: 1.5E2 berarti $1.5 * 10^2$.
2. Dalam notasi algoritmik, operator relasional kesamaan tidak berlaku untuk bilangan riil. Untuk harga riil, harus didefinisikan suatu bilangan kecil ε yang menyatakan ketelitian perhitungan, termasuk yang disebut dengan "kesamaan".
3. Pada bahasa pemrograman yang nyata, operator kesamaan bilangan riil **mungkin** dapat dipakai dengan “baik”, namun harus hati-hati dan sangat spesifik untuk implementasi bahasa tertentu. Dengan alasan ini, pada notasi algoritmik operator kesamaan bilangan riil dianggap tidak ada.

Karakter

Nama : **character**
Domain : **Himpunan** yang terdefinisi oleh suatu enumerasi, misalnya:
```
['0'..'9','a'..'z','A'..'Z',, RETURN, SPACE, EOL]
```

Ada karakter yang kelihatan dan tidak kelihatan, dituliskan dengan “nama” seperti pada contoh (RETURN, SPACE, EOL).
Ada keterurutan (suksesor dan predesesor), yang ditentukan oleh representasi di dalam komputer, misalnya pengkodean ASCII.

Konstanta : Dituliskan di antara tanda petik atau suatu nama
`'A'  'P'  'K'  RETURN  'M'`

Operator :

KELOMPOK	Op	ARTI	Hasil
Operator perbandingan	=	Sama dengan	`boolean`
	≠	Tidak sama dengan	`boolean`

String

Ada sebuah type yang sangat diperlukan di hampir semua sistem, yang pada akhirnya dapat dianggap "setengah" type dasar karena sudah tersedia, yaitu **String**. Untuk selanjutnya, type string dapat dianggap type primitif.

Untuk membedakan string dengan karakter, sebagai pembatas digunakan tanda petik ganda (").

Contoh:

Type : string
Domain : untaian karakter yang didefinisikan pada domain character
Konstanta : `"KU"  "Anak hilang"  "Pisang"  "K"  ""`
Operator :

KELOMPOK	Op	ARTI	Hasil
Operator perbandingan	=	Sama dengan	`boolean`
	≠	Tidak sama dengan	`boolean`
Konstruksi	•	Tambah satu karakter di akhir string string x character → string	`string`
	o	Tambah satu karakter di awal string character x string → string	`string`
	&	konkatenasi string x string → string	`string`

Type Enumerasi

TYPE enumerasi adalah type yang tidak definisi domainnya tidak dilakukan menurut suatu aturan (*by definition*) melainkan dengan melakukan "enumerasi" atau menyebutkan satu per satu nilai anggotanya. Type enumerasi mewakili himpunan nilai yang diberi nama.

Karena disebutkan satu per satu, dalam suatu type enumerasi biasanya dikenal cara akses suatu nilai anggota enumerasi lewat katakunci sebagai berikut :

- First, yaitu anggota nilai yang pertama
- Last, yaitu anggota nilai yang terakhir
- Successor (elemen) yaitu anggota nilai yang berikutnya dari elemen
- Predesesor(elemen), yaitu anggota nilai yang sebelumnya dari elemen

Type ini seringkali tersedia dalam bahasa tingkat tinggi, gunanya untuk mendefinisikan dengan lebih jelas suatu himpunan nilai yang pasti, misalnya:
type hari : (senin, selasa, rabu, kamis, jumat, sabtu, minggu)
type warna : (merah,kuning, hijau, biru, nila, ungu)

Contoh: Type hari
Type **hari** menyatakan enumerasi nama hari dalam minggu

```
type hari : (senin, selasa, rabu, kamis, jumat, sabtu, minggu)
```

Jika dideklarasi NAMA variabel H sebagai berikut:

```
H : hari { artinya: H adalah nama bertype "hari" }
```

Maka cara mengacu/mengakses nilai elemen yang tersimpan pada P yang telah terdefinisi adalah:

```
First (H)          { menghasilkan nilai: senin }
Last (H)           { menghasilkan nilai: minggu }
Succ (selasa)      { menghasilkan nilai: rabu }
Prec (selasa)      { menghasilkan nilai: senin }
```

Domain : **nilai yang dienumerasi**, yaitu:
senin, selasa, rabu, kamis, jumat, sabtu, minggu
Konstanta : senin sabtu

Type Bentukan

Type bentukan adalah suatu TYPE yang dirancang/dibentuk (dan diberi nama) dari beberapa komponen bertype tertentu, jadi merupakan sekumpulan elemen bertype dasar atau bertype yang sudah dikenal. Type bentukan dibuat/didefinisikan karena perancang program memutuskan bahwa keseluruhan (hasil komposisi) komponen type tersebut mempunyai sebuah makna semantik Ada relasi yang persis antara satu elemen dengan yang lain. Operasi terhadap komponen (elemen) bertipe dasar dilakukan seperti yang didefinisikan pada tipe dasar. Operasi terhadap keseluruhan tipe mungkin didefinisikan atau tidak.

Type bentukan seringkali disebut sebagai type komposisi, agregat. Implementasinya dalam suatu bahasa sangat bervariasi satu sama lain. Dalam notasi algoritmik, sebuah type bentukan berupa agregasi elemen dituliskan dengan notasi :

```
type namatype : < elemen1 : type1,
                  elemen2 : type2,
                  ...   >
```

Perhatikan dalam pengertian sebagai type bentukan, maka ada keseluruhan type yang harus dibentuk menurut pembentuk tertentu yaitu Konstruktor, atau adanya komponen type yang harus dapat diacu oleh Selektor. Operator terhadap type tersebut harus dibuat. Hal ini akan dibahas lebih mendalam dan terstruktur dalam kuliah Struktur Data karena pada hakekatnya, membentuk sebuah type berarti menentukan Struktur Data.

Pada bagian ini, hanya diberikan contoh-contoh dan yang diutamakan adalah notasi tentang bagaimana mengakses elemen type oleh notasi akses yang tersedia.

Contoh 1: Type Point
Type **Point** menyatakan absis dan koordinat real pada sumbu kartesian

```
type Point : <  X : real, { absis }
                Y : real  { ordinat }
             >
```

Jika dideklarasi nama variabel P sebagai berikut:

```
P : Point  { artinya: P adalah sebuah Point }
```

Maka cara mengacu/mengakses nilai elemen yang tersimpan pada P yang telah terdefinisi adalah:

```
P.X   { menghasilkan nilai absis bertype real }
P.Y   { menghasilkan nilai ordinat bertype real }
```

Domain : <real, real>
Konstanta : <5.0,6.0> <6.0,100.0>
Operator :
- operator terhadap Point harus dibuat.
- operasi real terhadap P.X dan P.Y.

Contoh 2: Type Jam (Versi 1)
Type **Jam** menyatakan representasi "jam" dalam notasi HH:MM:SS dengan HH bernilai [0..23]; MM bernilai [0..59] dan SS bernilai [0..59] }

```
type Jam : < HH : integer [0..23], { jam }
             MM : integer [0..59], { menit }
             SS : integer [0..59]  { detik }
           >
```

Jika dideklarasi nama variabel J sebagai berikut:

```
J : Jam  { artinya : J adalah sebuah JAM }
```

Maka cara mengacu/mengakses nilai elemen yang tersimpan pada P yang telah terdefinisi adalah:

```
J.HH  {menghasilkan nilai bagian jam bertype integer[0..23]}
J.MM  {menghasilkan nilai bagian menit bertype integer[0..59]}
J.SS  {menghasilkan nilai bagian detik bertype integer[0..59]}
```

Domain : <integer, integer, integer>
Konstanta : <0,0,0> <15,20,30>
Operator :
- operator terhadap JAM harus dibuat.
- operasi integer terhadap komponen HH, MM, SS.

Contoh 3: Type Jam (Versi 2)

Type **Jam** menyatakan representasi "jam" dalam notasi HH:MM:SS dengan HH bernilai [0..11], MM bernilai [0..59], SS bernilai [0..59], dan ampm yang merupakan enumerasi (am, pm)

```
type Jam : < HH : integer [0..11], { jam }
            MM : integer [0..59], { menit }
            SS : integer [0..59], { detik }
            ampm : (am, pm)       { menentukan siang, malam }
          >
```

Jika dideklarasi nama variabel J sebagai berikut :

```
J : Jam  { artinya : J adalah sebuah JAM }
```

Maka cara mengacu/mengakses nilai elemen yang tersimpan pada P yang telah terdefinisi adalah:

```
J.HH  {menghasilkan nilai bagian jam bertype integer[0..11]}
J.MM  {menghasilkan nilai bagian menit bertype integer[0..59]}
J.SS  {menghasilkan nilai bagian detik bertype integer[0..59]}
J.ampm {menghasilkan nilai am atau pm }
```

Domain : <integer, integer, integer, (am,pm)>
Konstanta : <0,0,0,am> <15,20,30,pm >
Operator :

- operator terhadap JAM harus dibuat.
- operasi integer terhadap komponen HH, MM, SS.

Contoh 4: Sistem untuk Penjadwalan di ITB

```
type Jam        : integer [11..610]   { lihat catatan }
type Dosen      : string
type Matakuliah : string
type Kelas      : integer [1..9999]
```

Type terstruktur **Jadwal**:

```
type  Jadwal :  < J : Jam,
                  D : Dosen,
                  MK : Matakuliah,
                  KL: Kelas >
```

Jika dideklarasikan sebuah nama **Jadwal_Kuliah : Jadwal**
Maka cara mengacu nilai yang tersimpan pada Jadwal_Kuliah adalah**:**

```
Jadwal_Kuliah.J   { integer[11..610] }
Jadwal_Kuliah.D   { string }
Jadwal_Kuliah.MK  { string }
Jadwal_Kuliah.KL  { integer[1..9999] }
```

Domain : sesuai dengan domain masing-masing komponen
Konstanta : <15,"Mary","IF221",9012>
<61,"Christine","IF223",100>
Operator :

- untuk Jadwal, tidak terdefinisi operator
- tapi kita dapat mengadakan:
  - operasi integer terhadap Jadwal.J asal nilainya [11..610].
  - operasi string terhadap Jadwal.D.
  - operasi string terhadap Jadwal.MK.
  - operasi integer terhadap Jadwal.KL masih dalam domain [1..9999].

Catatan :
Jam perkuliahan kuliah di ITB dikode: 11, 12, ..., 19, 110, 21, 22, ..., 29, 210, ..., 61, 62, ..., 69, 610. Jika dikehendaki mendefinisikan dengan lebih teliti, maka domain harga dapat dibuat lebih persis dengan enumerasi nilai:
[11..19, 110, 21..29, 210, 31..39, 310, 41..49, 410, 51..59, 510, 61..69, 610]

Contoh 5: Type Bentukan

```
type Dosen : string
type Matakuliah : string
type Kelas : integer [11..9999]
type ProgramStudi : string  ["EL","MS","TI","TK","TF","IF"]
type Jam : integer [11..610]
type DosenMK : < D : Dosen, MK : Matakuliah >
```

Type terstruktur **Kuliah**:

```
type Kuliah : < Dari, Ke : Jam,
                DMK : DosenMK,
                PS : ProgramStudi >
```

Cara mengacu nilai yang tersimpan pada **Kuliah_ITB** yang bertype **Kuliah** adalah:

```
Kuliah_ITB.Dari          { integer [11..610] }
Kuliah_ITB.Ke            { integer [11..610] }
Kuliah_ITB.DMK           { DosenMK}
Kuliah_ITB.DMK.D         { string }
Kuliah_ITB.DMK.MK        { string }
Kuliah_ITB.PS            { string }
```

Domain : sesuai dengan domain masing-masing komponen

Konstanta :
```
<15,16,<"Marni",9012>,"IF">
<21,22,<"Edi",9012>,"TF">
```

NILAI, EKSPRESI, INPUT, DAN OUTPUT

Komputer mampu melakukan operasi aritmetika dan logika terhadap nilai yang disimpan di memori. Di dalam program, nilai disimpan dalam suatu nama "variabel", sehingga dengan mengacu kepada nama, dapat dilakukan operasi yang diinginkan. Nilai yang disimpan juga dapat diperoleh dari hasil pembacaan dari piranti masukan, serta dapat dikomunikasikan ke dunia luar melalui piranti keluaran.

Pada bagian ini dijelaskan notasi yang dipakai untuk mendefinisikan dan melakukan manipulasi nilai.

Nilai (Harga)

Nilai atau harga adalah suatu besaran bertype yang telah dikenal.

Harga dalam suatu algoritma dapat diperoleh dari :
- **isi suatu nama**, yaitu nama informasi atau nama konstanta,
- hasil perhitungan suatu **ekspresi,**
- hasil yang dikirim suatu **fungsi**,
- **konstanta** bernama atau tanpa diberi nama yang dipakai langsung.

Harga dapat **dimanipulasi**:
- diisikan ke **nama** informasi (nama variabel) yang mempunyai type sesuai dengan harga tersebut dengan instruksi "*assignment*",
- diacu saja dari suatu nama, untuk dipakai dalam perhitungan atau ekspresi,
- dibandingkan, sesuai dengan operator pembanding yang tersedia,
- dituliskan ke piranti keluaran (layar, printer, menyalakan signal, ...),
- dipakai dalam **ekspresi**, tergantung typenya.

Pengisian Nilai

Suatu nama konstanta secara otomatis akan mempunyai harga tetap yang terdefinisi pada saat nama konstanta tersebut didefinisikan dalam kamus. Jadi, menyebutkan nama konstanta secara otomatis akan memakai harga yang didefinisikan pada kamus tersebut. Tidak demikian halnya dengan nama informasi. Suatu nama informasi dapat dipakai dalam ekspresi jika harganya telah terdefinisi. Ada dua cara untuk mengisi suatu nama informasi dengan harga, yaitu dengan:
- *assignment*, atau
- dibaca dari suatu piranti masukan

Assignment

Assignment adalah instruksi primitif algoritmik untuk menyimpan harga pada suatu nama informasi yang isinya boleh bervariasi ("variabel"), dengan perkataan lain adalah memberikan harga pada suatu nama variabel. Dengan pemberian harga ini,

harga lama yang disimpan tidak lagi berlaku, yang berlaku adalah harga paling akhir yang diberikan.

Memrogram secara prosedural pada hakekatnya adalah memanipulasi nama yang mewakili alokasi memori tertentu dan memaipulasinya dengan algoritma yang ditulis. Manipulasi harga terhadap nama dilakukan dengan *assignment*.

```
ALGORITMA
   <nama1> ← <nama2>      { harga dari nama2 disalin ke nama2 }
                          { harga nama1 sama dengan harga nama2
                            setelah instruksi ini }

   <nama> ← <konstanta>   { harga konstanta diisikan ke nama }

   <nama> ← <ekspresi>    { hasil perhitungan ekspresi diisikan ke
                            nama }
```

dengan syarat:
- bagian kiri dan bagian kanan tanda pemberian harga (←) bertype sama,
- <nama> dan <nama1> (bagian kiri tanda ← harus merupakan **nama informasi**, tidak boleh nama konstanta, type, fungsi atau prosedur,
- nama yang tertulis di bagian kanan tanda ← (misalnya <nama2> atau nama <konstanta> atau nama yang dipakai dalam ekspresi) boleh berupa nama informasi, nama fungsi, nama konstanta,
- semua nama yang dipakai dalam *assignment* tidak boleh berupa nama type atau nama prosedur.

Input

Selain dengan *assignment*, suatu harga dapat diisikan ke suatu nama informasi melalui **pembacaan** harga tersebut dari piranti masukan (*keyboard*, *mouse*, *scanner*, dan sebagainya). Disebut "dibaca" karena arah dari pengisian harga yaitu seakan-akan komputer "membaca" harga yang diberikan pengguna. Pemberian harga dari piranti masukan ini mencakup konsep "menerima nilai" dari piranti masukan apapun, misalnya menerima nilai besaran temperatur dari sebuah sensor temperatur yang dihubungkan dengan komputer di suatu ruangan.

```
ALGORITMA
   input(list-nama)
```

dengan syarat:
- list nama adalah satu atau lebih **nama informasi**,
- nama yang muncul pada list-nama hanya boleh berupa nama **informasi**, dan tidak boleh nama lain (nama konstanta, type, fungsi atau prosedur).

Output

Suatu harga yang disimpan dalam memori komputer (diacu berkat definisi nama informasi (variabel), nama konstanta atau konstanta) harus dapat dikomunikasikan ke dunia luar untuk diinterpretasikan oleh pemakai program. Dalam hal ini, harga harus dapat **dituliskan** ke suatu piranti keluaran, misalnya layar, printer.

Instruksi algortimik yang disediakan untuk menuliskan nama informasi adalah instruksi **penulisan** atau ***output***. Instruksi output tidak mengubah nilai yang disimpan.

```
ALGORITMA
   output(<list-nama>)
            { Semua harga yang tersimpan dalam setiap nama yang ada
              pada list-nama akan dituliskan pada piranti keluaran
              sesuai dengan urutan penulisan nama. Perhatikan bahwa
              yang dituliskan ke piranti keluaran hanya harga yang
              disimpan saja. Lihat contoh. }

   output(<konstanta>)
            { Harga konstanta ke piranti keluaran }

   output(<ekspresi>)
            { Harga hasil perhitungan ekspresi dituliskan ke piranti
              keluaran }

   output(<list-nama>, <konstanta>, <ekspresi>)
            { Yang dituliskan ke piranti keluaran adalah semua harga
              sesuai dengan urutan penulisan nama, konstanta,
              ekspresi                                              }
```

dengan syarat :

- `<list-nama>` adalah satu atau lebih nama: boleh nama **informasi**, nama **konstanta** atau hasil pemanggilan/aplikasi **fungsi**. Khusus untuk pemanggilan fungsi, lihat pemakaian fungsi.
- Nama-nama dalam `<list-nama>` tidak boleh berupa nama **type** atau **prosedur**.
- Nama yang akan dituliskan sudah terdefinisi harganya. Jika suatu nama informasi, didefinisikan dengan ***assignment*** atau instruksi **input**.

Ekspresi

Ekspresi suatu “rumus perhitungan", yang terdiri dari operan dan operator. **Operator** yang dituliskan harus didefinisikan untuk mengoperasikan **operan** ber-**type** tertentu. Hasil perhitungan adalah **harga** dengan domain yang memenuhi **type** operator yang bersangkutan.

Operan harus mempunyai harga, karena itu dapat berupa **konstanta**, **nama** (dalam hal ini yang dipakai dalam perhitungan adalah harga yang dikandung nama yang bersangkutan), hasil pengiriman suatu **fungsi** atau merupakan suatu **ekspresi**.

Ekspresi **uner** adalah ekspresi dengan operator uner, yaitu operator yang hanya membutuhkan satu operan.

Ekspresi **biner** adalah ekspresi dengan operator biner (membutuhkan dua operan) dapat dituliskan dalam 3 macam notasi, yaitu:

a. Notasi **infix**: operator di tengah
   operan1 operator operan2
   Contoh:
   13 * 5
   ((3 * 5) + (4 div 7))- (a * b)

b. Notasi **prefix**: operator di awal
   operator operan1 operan2
   Contoh :
   * 13 5
   - + * 3 5 div 4 7 * a b adalah - ((+ (* 3 5) (div 4 7)) (* a b))

c. Notasi **suffix/polish**: operator di akhir
   operan1 operan2 operator
   Contoh :
   13 5 *
   3 5 * 4 7 div a b * + - adalah (3 5 *) ((4 7 div) (a b *) +) -

Untuk selanjutnya, pada kuliah ini ekspresi dituliskan dalam bentuk *infix*, yang sesuai dengan penulisan ekspresi aritmatika sehari-hari. Ada bahasa pemrograman memakai ekspresi infix, prefix atau postfix. Untuk menghindari kerancuan prioritas perhitungan, ekspresi ditulis dengan tanda kurung yang lengkap.

Ekspresi akan dihitung (dengan beberapa perjanjian jika terjadi ketidak-cocokan type maupun ketelitian). Hasilnya sesuai dengan **type ekspresi**, selanjutnya dapat dimanipulasi, ditampilkan pada piranti keluaran atau disimpan dalam suatu nama. Type ekspresi sesuai dengan type hasil. Contoh type ekspresi untuk type dasar adalah:
- logika (boolean)
- numerik
- karakter dan string

Contoh Ekspresi Boolean

Diberikan sebuah kamus dan algoritma untuk mendefinisikan nama variabel

```
KAMUS
{ Nama Konstanta }
   constant benar : boolean = true

{ Nama Informasi }
   Found : boolean
   Flag : boolean
```

```
ALGORITMA
{ Pada bagian ini telah didefinisikan sehingga nilai berikut:
  Harga Found adalah true dan Flag adalah false }
```

Ekspresi boolean	Hasil	Keterangan
true and false	false	
true or false	true	
benar xor true	false	
Flag	false	Flag = false
not Found	false	Found = true
Flag and Found	false	Flag = false, Found = true

Contoh Ekspresi Numerik

Diberikan sebuah kamus dan algoritma untuk mendefinisikan nama variabel:

```
KAMUS
{ Nama Konstanta }
   constant PI : real = 3.14159
   constant Faktor : integer = 3

{ Nama Informasi }
   i, j : integer
   Jum  : integer
   x, y : real
```

```
ALGORITMA
{ Pada bagian ini telah didefinisikan sehingga nilai berikut:
  Harga i adalah 5, j adalah 0, Jum adalah 4, x adalah 0.0,
  dan y adalah 7.5 }
```

Maka berikut ini adalah contoh ekspresi numerik dan hasilnya:

Ekspresi	Hasil	Keterangan
1 * 5	5	Ekspresi integer
1 + 3 * 5	16	Ekspresi integer
1 / 3	0.333	Ekspresi integer, hasil riil
11 > 3	true	Ekspresi relasional untuk integer
11 > 21	false	Ekspresi relasional untuk integer
10 mod 3	1	Ekspresi integer
10 div 3	3	Ekspresi integer
1+7 div 3	2	Ekspresi integer
1./3.	0.333	Ekspresi real
10./2.	5.	Ekspresi real
i + 1	6	i = 5, ekspresi integer
i + j	5	i = 5 dan j = 0, ekspresi integer
x * PI	0.0	x = 0.0 dan PI = 3.1415, ekspresi riil
i + x	??	Operan tidak sejenis
x + y	0.0	x = 0.0 dan y = 7.5, ekspresi riil

Contoh Ekspresi Karakter dan String

Diberikan sebuah kamus dan algoritma untuk mendefinisikan nama variabel:

```
KAMUS
{ Nama Konstanta }
   constant blank : character  = ' '

{ Nama Informasi }
   CC : character
   str1, str2 : string
```

```
ALGORITMA
{ Pada bagian ini telah didefinisikan sehingga nilai berikut:
  Harga CC adalah 'A', str1 adalah "AKU", dan str1 adalah "X" }
```

Ekspresi	Hasil	Keterangan
CC = 'X'	false	Ekspresi pembandingan
blank	' '	Nilai dari konstanta
str1 & str2	"AKUX"	Konkatenasi
CC o str1	"AAKU"	Operasi menambah karakter di awal
str1 • CC	"AKUA"	Operasi menambah karakter di akhir

Latihan Soal

Didefinisikan nama dalam kamus sebagai berikut:

```
KAMUS
   constant Pi : real = 3.1714
   GajiTotal, GajiPokok, Tunjangan, PotGaji : real
   MID1, MID2 : integer
   Urutan, ranking  : integer
   CC1, CC2 : character
   S1, Message : string
   Found : boolean
   function AddXY (X, Y : integer) → integer
   { Menghasilkan penjumlahan X+Y }
```

Berikut ini adalah contoh ekspresi dalam notasi infix untuk kamus dia atas. Periksalah apakah ekspresi yang dituliskan benar. Jika benar tentukan jenis hasilnya, jika salah koreksilah.

Ekspresi	Jenis hasil dan komentar
Urutan – 1	
CC1 or CC2	
Found and (Urutan>100)	
(Urutan ^ 2) +GajiTotal+ADDXY(MID1,MID2)	
GajiPokok + Tunjangan – PotGaji	
Urutan * 2 /ranking	
Message & S1	
Message = "HELLO"	

AKSI SEKUENSIAL

Aksi sekuensial (*sequential statement)* adalah struktur kontrol algoritmik yang paling sederhana. *Sequential statement* adalah sederetan instruksi primitif dan/atau aksi yang akan dilaksanakan (dieksekusi) oleh komputer berdasarkan urutan penulisannya. Jadi, jika dituliskan sebuah *sequential statements* yang terdiri dari deretan instruksi/aksi ke 1, 2, 3, 4, ..., n maka setiap instruksi/aksi akan dilaksanakan secara sekuensial mulai dari yang ke 1, kemudian ke-2, ke-3, ..., s.d. ke-n. Program paling sederhana tentunya hanya mengandung satu satu instruksi saja.

Initial state dari *sequential statement* adalah *state* awal yang harus dipenuhi dan ***Final state*** dari *sequential statement* adalah *final state* setelah instruksi/aksi terakhir. *Final state* dari sebuah instruksi/aksi yang ada pada urutan instruksi/aksi ke-i akan menjadi *initial state* dari instruksi/aksi ke-i+1. Dengan kata lain, urut-urutan penulisan instruksi/aksi pada suatu *sequential statement* sangat penting.

Notasi Algoritmik untuk Instruksi Sekuensial

Urutan penulisan instruksi/aksi pada *sequential statement* adalah sesuai dengan penulisannya per baris. Aksi sekuensial dapat juga dituliskan menjadi satu baris program dengan cara memisahkan penulisan setiap instruksi/aksi dengan tanda "titik koma". Penulisan aksi sekuensial dengan dipisahkan titik koma sebaiknya hanya dilakukan untuk aksi sekuensial yang jika urutan penulisannya diubah tidak berpengaruh kepada program

```
Program SEQ1
{ Contoh penulisan aksi sekuensial per baris }
KAMUS
    i : integer
    x : real
    flag : boolean

ALGORITMA
    input (i)
    x ← 0.0
    flag ← true
    output (x)
    output (i*2, flag)
```

```
Program SEQ2
{ Contoh penulisan aksi sekuensial dengan tanda titik koma}
KAMUS
    i : integer
    x : real
    flag : boolean

ALGORITMA
    input (i); x ← 0.0;  flag ← true
    output (x); output (i*2, flag)
```

Ada aksi sekuensial yang jika diubah urutan instruksi/aksinya akan mempengaruhi eksekusi program. Ada *sequence* yang jika diubah urutan instruksi/aksinya akan menghasilkan efek neto yang sama (tidak berpengaruh).

Contoh aksi sekuensial yang berpengaruh jika diubah urutannya:

```
Program SEQ3
{ Aksi sekuensial yang berpengaruh jika diubah urutannya }

KAMUS
    i : integer

ALGORITMA
{ Jika urut-urutan dua buah instruksi pada program dengan dua buah instruksi
sekuensial sebagai berikut diubah, maka akan menghasilkan kesalahan fatal
karena syarat untuk dapat menuliskan harga yang tersimpan di suatu nama
adalah bahwa nama tersebut sudah terdefinisi harganya }

{ State : i belum terdefinisi }
   input (i)
{ State : i terdefinisi akibat instruksi input }
   output (i)
{ State : i tertulis di piranti keluaran }
```

```
Program SEQ3
{ aksi sekuensial yang berpengaruh jika diubah urutan instruksinya }

KAMUS
    i : integer
    j : integer

ALGORITMA
{ Jika urut-urutan instruksi dibalik, program akan salah. Nilai j harus
diisi dengan i yang terdefinisi dari pembacaan; penulisan I dan j hanya
dapat dilakukan jika sudah terdefinisi }

{ State : harga i dan j tidak terdefinisi }
   input (i)
{ State : i terdefinisi }
    j ← i
{ State : harga j terdefinisi, disalin dari i,
          harga i dan j terdefinisi }
  output (i,j)
{ State : harga i dan j tertulis di piranti keluaran }
```

Pada bagian ini akan diberikan beberapa contoh program yang hanya mengandung aksi sekuensial dan hanya mempergunakan instruksi yang pernah dipelajari sebelumnya yaitu manipulasi nama dan harga.

Jika bagian prosedur, analisis kasus dan pengulangan serta yang lain telah dipelajari, maka aksi sekuensial boleh mengandung analisis kasus dan pengulangan serta aksi yang yang dinyatakan dengan nama prosedur.

Contoh 1: HELLO

Persoalan:
Tuliskanlah algoritma untuk menulis "HELLO" ke piranti keluaran yang disediakan. Berikut ini diberikan 2 solusi. Pikirkanlah, mana yang lebih "baik".

Spesifikasi:
Input : -
Output : "HELLO"
Proses : menulis "HELLO"

```
Program HELLO1
{ Menulis "HELLO" ke piranti keluaran }
KAMUS
ALGORITMA
   output ("HELLO")
```

```
Program HELLO2
{ Menulis "HELLO" ke piranti keluaran }
KAMUS
   pesan : string  { nama informasi yang dituliskan pesannya }
ALGORITMA
   pesan ← "HELLO"
   output (pesan)
```

Contoh 2: HELLOX

Persoalan:
Tuliskanlah algoritma untuk membaca sebuah nama, dan menulis "HELLO" yang diikuti dengan nama yang diketikkan. Contoh :

jika dibaca "ALI", maka keluaran adalah : "HELLO ALI"
jika dibaca "SINTA", maka keluaran adalah : "HELLO SINTA"

Spesifikasi:
Input : nama
Output : "HELLO <nama>"
Proses : menulis "HELLO" diikuti nama yang dibaca

```
Program HELLOX
{ Menulis "HELLO" berikut nama yang diberikan dari piranti masukan ke
piranti keluaran }
KAMUS
   name : string  { nama informasi yang dituliskan pesannya }
ALGORITMA
   input (name)
   output ("HELLO ", name)
```

Contoh 3: JARAK

Persoalan:
Dibaca dua buah harga v (kecepatan, m/detik) dan t (waktu, detik), yang mewakili koefisien persamaan gerak lurus beraturan. Harus dihitung dan dituliskan hasilnya, jarak yang ditempuh benda yang bergerak lurus beraturan dengan kecepatan v tersebut dalam waktu t.

Spesifikasi:
Input : v (kecepatan, m/detik), integer dan t (waktu, detik), integer
Proses : menghitung S = v * t
Output : S (jarak yang ditempuh dalam meter), integer

```
Program JARAK1
{ Dibaca v dan t, Menghitung S = v * t  dan menuliskan hasilnya, }
{ dengan memakai nama antara }

KAMUS
   v : integer   { kecepatan, m/detik }
   t : integer   { waktu, detik }
   S : integer   { jarak (m) yang ditempuh dalam waktu t,
                   dengan kecepatan v, pada gerak lurus beraturan }

ALGORITMA
   input (v,t)
   S ← v * t
   output (S)
```

```
Program JARAK2
{ Dibaca v dan t, menghitung  jarak = v * t  dan menuliskan hasilnya }
{ Tanpa memakai nama antara }

KAMUS
   v : integer   { kecepatan, m/detik }
   t : integer   { waktu, detik }

ALGORITMA
   input (v,t)
   output (v*t)
```

Catatan:
1. Mengenai input yang dibaca oleh program: dalam kehidupan sehari-hari, akan sangat sulit untuk memasukkan input nilai v dan t di atas, karena jika pemakai program salah memasukkan urutan, harga yang tersimpan akan salah. Untuk contoh di atas, karena rumusnya hanya perkalian dua buah nilai integer mungkin tidak fatal. Jika rumus misalnya adalah v-t, maka pemasukan data dengan urutan yang lain akan fatal akibatnya.
2. Mengenai output: output program di atas sangat sulit diinterpretasi karena yang dihasilkan oleh program hanya sebuah angka yang tidak jelas interpretasinya.

```
Program JARAK3
{ Dibaca v dan t, menghitung  jarak = v * t  dan menuliskan hasilnya }
{ Tanpa memakai nama antara }

KAMUS
    v : integer   { kecepatan, m/detik }
    t : integer   { waktu, detik }

ALGORITMA
    output ("Input nilai kecepatan : ")
    input (v)
    output ("Input nilai waktu : ")
    input (t)
    output ("Jarak yang dihitung = ", v * t)
```

Catatan:

1. Dengan menuliskan semacam ini, program akan dapat dioperasikan dengan lebih mudah, namun algoritma menjadi sangat rinci. Lebih rinci lagi, dapat dibuat layar yang indah dengan warna-warni dan posisi penulisan yang enak dibaca.
2. Tujuan dari menuliskan algoritma adalah untuk menuliskan "sketsa" solusi dari program, jadi hanya mengandung hal yang esensial.
3. Sebaiknya instruksi yang sudah sangat rinci dan tidak mengandung hal esensial dikode secara langsung dalam bahasa pemrograman pada saat impelementasi. Jadi, teks algoritma JARAK1 dan JARAK2 yang hanya mengandung hal yang esensial adalah produk dari design program, sedangkan teks rinci semacam JARAK3 langsung pada bahasa pemrogramannya.
4. Untuk selanjutnya, teks algoritma dituliskan dengan hanya mengandung hal yang esensial saja karena pusat perhatian kita adalah untuk menghasilkan sketsa solusi saja.

Contoh 4: JAMMENITDETIK

Persoalan:

Dibaca sebuah harga berupa bilangan bulat, positif dan lebih kecil dari 1 juta, yang mewakili besaran dalam detik. Harus dihitung ekivalensinya, berapa hari, jam berapa menit dan berapa detik. Contoh: data 309639 akan menghasilkan 3, 14, 0, 39, yang artinya 3 hari, 14 jam, 0 menit dan 9 detik.

Spesifikasi:

Input : n (detik), integer
Proses : menghitung hari, jam, menit, detik ekivalen dengan n
Output : HARI, JAM, MENIT, DETIK

Analisis: nama-nama informasi yang akan dibutuhkan adalah:
n : bilangan yang dibaca sebagai data, integer antara 0 dan 999999
H : HARI, bilangan bulat positif, HARI
J : JAM, bilangan bulat positif antara 0 - 23
M : MENIT, bilangan bulat antara 0 -59
D : DETIK, bilangan bulat antara 0 – 59
Rumus : 1 hari = 86400 detik; 1 jam = 3600 detik dan 1 menit = 60 detik.

```
Program JAMMENITDETIK
{ Dibaca  n (integer), besaran dalam detik}
{ Harus dihitung J (Jam) dan M(Menit) dan D(Detik sisanya), dan dituliskan
hasilnya }
```

```
KAMUS
   n  : integer [0..999999] { data yang dibaca }
   H  : integer ≥ 0         { HARI, bilangan bulat positif }
   J  : integer [0..23]     { JAM, bilangan bulat positif antara 0 - 23 }
   M  : integer [0..59]     { MENIT, bilangan bulat antara 0 -59 }
   D  : integer [0..59]     { DETIK, bilangan bulat antara 0 - 59 }
   rH : integer [0..86399]  { sisa detik dalam perhitungan HARI }
   rJ : integer [0..3599]   { sisa detik dalam perhitungan JAM }
```

```
ALGORITMA
   input (n) { 0 ≤ n ≤ 999999 }
   H ← n div 86400; rH ← n mod 86400
   { n = 86400*H + rH and 0 ≤ rH < 86400 }
   J ← rH div 3600; rJ ← rH mod 3600
   { n = 86400*H + 3600*J + rJ and 0 ≤ rH < 86400 and  0 ≤ rJ < 3600 }
   M ← rJ div 60; D ← rJ mod 60
   { n = 86400*H + 3600*J + 60*M + D and
     0 ≤ rH < 86400 and 0 ≤ rJ < 3600 and 0 ≤ M < 60 and 0 ≤ D < 60 }
   output (H,J,M,D)
```

Catatan:

1. Ini adalah contoh program dengan asersi yang lengkap, yang merupakan pembuktian kebenaran program. Namun setiap baris mengandung asersi. Untuk selanjutnya, asersi dituliskan "secukupnya", dan pelajaran mengenai asersi yang lengkap akan dicakup pada matakuliah "Analisis Algoritma".
2. Program tersebut adalah pola solusi untuk "menguraikan" sebuah besaran (detik) menjadi besaran lain (hari, jam, menit, detik) berdasarkan rumus konversi besaran.

Pemilihan nama informasi pada program tersebut kurang mengandung artinya, maka disertakan definisi nama singkat sebagai komentar. Jika pemilihan nama sudah interpretatif, tidak perlu lagi deskripsi nama seperti pada contoh.

Contoh 5: KALKULASI TYPE TERSTRUKTUR PECAHAN

Persoalan:

Tuliskanlah algoritma untuk membaca dua buah besaran bertype pecahan, dan menuliskan hasil kali kedua pecahan tersebut. Pecahan harus direpresentasi sebagai dua buah bilangan integer yang menyatakan pembilang dan penyebut. Untuk penyederhanaan, penyebut selalu tidak pernah sama dengan nol. Pecahan negatif ditandai dengan pembilang berupa integer negatif

Contoh

Pecahan <1,2> merepresentasi 1/2
Pecahan <-4,2> merepresentasi -4/2
Pecahan <1,1> merepresentasi 1/1
Pecahan <0,2> merepresentasi 0/2
Pecahan <1,0> bukan pecahan, karena di luar definisi pecahan

```
Program PECAHAN1
{ Input : P1, P2 bertype pecahan;
  Output : hasil perkalian P1 dan P2 }
{ Membaca dua buah pecahan, mengalikan serta menuliskan hasilnya }
{ Tanpa nama antara }
KAMUS
    type Pecahan : < Pembilang : integer, Penyebut : integer > 0 >
    P1 : Pecahan
    P2 : Pecahan
ALGORITMA
    input (P1,P2)
    output ( < P1.Pembilang*P2.Pembilang, P1.Penyebut*P2.Penyebut > )
```

```
Program PECAHAN2
{ Input : P1, P2 bertype pecahan;
  Output : hasil perkalian P1 dan P2 }
{ Proses : membaca dua buah pecahan, mengalikan serta menuliskan hasilnya }
{ Dengan memakai nama antara }
KAMUS
    type Pecahan : < Pembilang : integer, Penyebut : integer >
    P1 : Pecahan
    P2 : Pecahan
    PKali : Pecahan
ALGORITMA
    input (P1, P2)
    PKali.Pembilang ← P1.Pembilang *P2.Pembilang
    PKali.Penyebut ← P1.Penyebut*P2.Penyebut
    output (PKali)
```

Contoh-6 : KALKULASI TYPE TERSTRUKTUR JAM

Persoalan:
Tuliskanlah algoritma untuk membaca dua buah besaran bertype Jam yang mewakili awal dan akhir suatu percakapan telpon dan menuliskan durasi waktu dalam detik yang dihitung dari kedua jam yang dibaca. Type Jam terdiri dari Jam, menit dan detik dan dipakai sistem jam dengan jam 00:00:00 sampai dengan 24.60:60

Contoh :
Jam <12:00:00> mewakili jam 12 siang
Jam <00:00:00> mewakili jam 12 malam
Jam <23:10:00> mewakili jam 11:10:00 malam
Jam <12:60:00> BUKAN JAM !
Jam <25:00:00> BUKAN JAM !

```
Program DURASI
{ Input :  JamAwal, JamAKhir  bertype Jam (HH:MM:DD);
           dan JamAkhir SELALU ≥ JamAwal
  Output : Selisih waktu dalam detik antara JamAwal dan JamAkhir  }

KAMUS
   type  Jam : < HH : integer [0..23],
                 MM : integer [0..59],
                 DD : integer [0..59]  >
   JamAwal, JamAKhir : Jam
   Durasi : integer ≥ 0

ALGORITMA
   input (JamAwal, JamAkhir)
   Durasi  ← (JamAkhir.HH * 3600 + JamAkhir.MM * 60 + JamAkhir.DD) -
             (JamAwal.HH * 3600 + JamAwal.MM * 60 + JamAwal.DD)
   output (Durasi)
```

Catatan:
Ini adalah pola program, yang menunjukan cara perhitungan dengan komputer yang berbeda dengan cara manual sebagai berikut:

13 10 50	13 10 10
12 05 10 -	12 50 30 -
01 05 40	00 19 40

Cara manual akan membutuhkan algoritma yang lebih rumit!
Seringkali komputasi dengan cara konversi ke nilai absolut semacam ini dilakukan dengan komputer.

Latihan Soal

1. Apa komentar anda mengenai pemilihan NAMA-NAMA pada algoritma JAMMENITDETIK di atas?
2. Seorang programmer menuliskan pernyataan sebagai berikut, setelah mendefinisikan semua nama yang dipakai sebagai real:
   DUA=TUJUH + LIMA
   Cinta = Toto + Tita
   Apa komentar anda ?
3. Jika type jam diubah sehingga aturan penulisannya bukan lagi dalam domain 00:00:00 sampai dengan 23:59:59 melainkan menjadi dari 00:00:00 am sampai dengan 11:59:59 pm, apa yang harus dilakukan dengan program DURASI?
4. Tuliskanlah minimal 20 rumus dalam bidang fisika dan matematika yang dapat diprogram dengan program JARAK sebagai "pola" program.

Buatlah spesifikasi dan algoritma untuk persoalan-persoalan berikut:

1. Dibaca dua buah harga yang dihasilkan dari pengukuran Arus (Ampere) dan Tahanan (Ohm), harus dihitung tegangan yang dihasilkan.
2. Dibaca sebuah bilangan bulat (rupiah) yang positif, harus dihitung ekivalensinya dalam dollar ($) dan dituliskan hasilnya. Bagaimana dengan perubahan kurs yang sering terjadi?

3. Apa yang harus diubah jika misalnya selain menghitung ekivalensi dalam $ juga harus dihitung ekivalensi dalam Yen, DM dan FF?
4. Dibaca sebuah besaran riil, yang mewakili hasil pengukuran temperatur dalam derajat Celcius. Hitung ekivalensinya dalam derajat Fahrenheit, Rheamur dan Kelvin.
5. Dibaca nama dan jam kerja pegawai, harus dihitung honor pegawai tersebut jika upahnya perjam adalah Rp. 5000,- Perhatikan bahwa upah per jam setiap pegawai tidak sama, dan perubahan upah tidak sesering perubahan kurs.
6. Dibaca tiga buah bilangan bulat yang mewakili tiga buah tahanan dalam Ohm: R1, R2, dan R3, harus dihitung dan dituliskan tahanan total yang dihasilkan jika ketiganya dipasang seri/paralel.
7. Dibaca lima buah bilangan bulat A1, A2, A3, A4, dan A5, harus dihitung jumlahnya dan dituliskan hasilnya. Bagaimana jika yang dibaca adalah 1000 buah bilangan?

ANALISIS KASUS

Analisis kasus, yang melahirkan instruksi kondisional, adalah elemen primitif pembangun algoritma, yaitu memungkinkan kita untuk membuat teks yang sama namun menghasilkan eksekusi yang berbeda-beda.

Mendefinisikan analisis kasus adalah mendefinisikan :
- ***kondisi***, yang berupa suatu ekspresi yang menghasilkan true atau false,
- ***aksi*** yang akan dilaksanakan jika *kondisi* yang dipasangkan dengan *aksi* yang bersangkutan dipenuhi.

Konstruksi dari suatu analisis kasus dapat dimulai dari menentukan semua kondisi yang mungkin (dengan melakukan partisi domain), atau dimulai dari menentukan variasi aksi. Tidak ada rumus yang baku tentang bagaimana memulai menuliskan analisis kasus. Pada contoh-contoh yang diberikan, ada yang berangkat dari kondisi, dan ada yang dimulai dari menentukan aksi.

Notasi algoritmik secara umum untuk analisis kasus yang umum (banyak kasus):

```
depend on (<nama-nama>)
   <kondisi-1> : <aksi-1>
   <kondisi-2> : <aksi-2>
   <kondisi-3> : <aksi-3>
   ...
   <kondisi-N> : <aksi-N>
```

dengan syarat :
- `<kondisi-1>,<kondisi-2>,<kondisi-3>,...,<kondisi-N>` berdomain harga **[true,false]**.
- `<kondisi-1>,<kondisi-2>,<kondisi-3>,...,<kondisi-N>` adalah ekspresi lojik/boolean yang mengandung `<nama-nama>` sebagai operan.
- `<kondisi-1> ∩ <kondisi-2> ∩ <kondisi-3> ∩ ... ∩ <kondisi-N> = Ø`. Berarti semua kondisi *disjoint*, tidak ada kasus yang sama tercakup pada dua buah kondisi.
- `<kondisi-1> ∪ <kondisi-2> ∪ <kondisi-3> ∪ ... ∪ <kondisi-N> = U`. Berarti kondisi mencakup semua kemungkinan.

Jika hanya ada satu kasus yang mengakibatkan aksi, atau dua kasus komplementer, dapat dipakai notasi sebagai berikut :

SATU KASUS:

```
if (<kondisi>) then
    <aksi>
```

Jika <kondisi> benar, maka <aksi> dilakukan. Jika <kondisi> tidak benar, maka tidak terjadi apa-apa (efek neto "kosong").

DUA KASUS KOMPLEMENTER:

```
if (<kondisi>) then
   <aksi-1>
else     { not <kondisi> }
   <aksi-2>
```

Catatan:

1. Perhatikan "indentasi" penulisan. Pada diktat ini, "end" sengaja tidak dituliskan supaya mahasiswa memperhatikan indentasi penulisan.
2. Hati-hati dalam memakai "else" yang berarti kondisi implisit yang merupakan negasi dari kondisi. Penulisan kondisi "else" secara eksplisit sangat disarankan.

Contoh-1 : MAKSIMUM DUA HARGA

Persoalan:

Dibaca dua buah harga a dan b, a mungkin sama dengan b. Harus dituliskan harga yang lebih besar. Eksekusi akan menghasilkan dua kemungkinan: menuliskan a, jika a ≥ b, atau menuliskan b, jika a < b. Tidak mungkin keduanya.

Spesifikasi:

Input : a dan b integer
Proses : menuliskan harga yang lebih besar, dengan spesifikasi bahwa menuliskan a, jika a ≥ b, atau menuliskan b, jika a < b
Output : a atau b, integer

```
Program MAXAB
{ Dibaca nilai a dan b, menuliskan a jika a ≥ b, menuliskan b jika a < b }

KAMUS
   a, b : integer

ALGORITMA
   input (a,b)
   depend on (a,b)
      a ≥ b : output (a)
      a < b : output (b)
```

Contoh 2: WUJUD AIR

Persoalan:

Dibaca sebuah harga berupa bilangan bulat, yang mewakili pengukuran suhu air (dalam ^{O}C) pada tekanan atmosfir, harus dituliskan wujud air pada temperatur dan tekanan itu.

Spesifikasi:

Input : T (integer)
Proses : menuliskan wujud air sesuai dengan nilai T
Output : "Beku" jika $T \leq 0$
"Cair" jika $0 < T \leq 100$
"Uap" jika $T > 100$

```
Program WUJUDAIR1
{ Dibaca T(integer), temperatur air (dalam °C) pada tekanan atmosfir }
{ Harus dituliskan wujud air pada temperatur T: Beku, Cair atau Uap }

KAMUS
   T : integer

ALGORITMA
   input (T)
   depend on (T)
      T ≤ 0        : output ("Beku")
      0 < T ≤ 100  : output ("Cair")
      T > 100      : output ("Uap")
```

```
Program WUJUDAIR2
{ Dibaca  T(integer), temperatur air (dalam °C) pada tekanan atmosfir }
{ Akan dituliskan wujud air pada temperatur T, dengan memisahkan kasus
peralihan sebagai kasus khusus }

KAMUS
   T : integer

ALGORITMA
   input (T)
   depend on (T)
      T < 0        : output ("Beku")
      T = 0        : output ("Beku-Cair")
      0 < T < 100  : output ("Cair")
      T > 100      : output ("Uap")
      T = 100      : output ("Cair-Uap")
```

Catatan:

1. Solusi pada versi-1 berbeda dengan versi-2. Ini menyangkut spesifikasi dan batasan program yang harus dijabarkan dan disetujui bersama dengan pemesan program
2. Solusi versi-1 akan dapat dipakai untuk Temperatur yang bertype riil, namun solusi versi-2 akan menimbulkan masalah karena operator kesamaan tidak dapat dipakai untuk bilangan riil. Sebagai latihan, disarankan untuk menuliskan solusi versi-2 jika Temperatur direpresentasi sebagai bilangan riil.

Contoh 3: RANKING

Persoalan :

Dibaca tiga buah harga a,b dan c, harus dituliskan secara terurut, mulai dari yang terkecil sampai dengan yang terbesar. Ketiga bilangan yang dibaca selalu berlainan harganya.

Spesifikasi:

Input : a,b,c, tiga besaran integer
Proses : menuliskan harga yang dibaca mulai dari yang terkecil s/d yang terbesar
Output : a,b,c jika a < b dan b < c
a,c,b jika a < c dan c < b
b,a,c jika b < a dan a < c
b,c,a jika b < c dan c < a
c,a,b jika c < a dan a < b
c,b,a jika c < b dan b < a

```
Program RANKING1
{ Dibaca tiga harga a, b, dan c, a ≠ b, b ≠ c, dan a ≠ c }
{ Harus dituliskan dari yang terkecil sampai dengan yang terbesar }

KAMUS
   a, b, c : integer

ALGORITMA
   input (a,b,c)
   depend on (a,b,c)
      a < b < c : output (a,b,c)
      a < c < b : output (a,c,b)
      b < a < c : output (b,a,c)
      b < c < a : output (b,c,a)
      c < a < b : output (c,a,b)
      c < b < a : output (c,b,a)
```

```
Program RANKING2
{ Dibaca tiga harga a, b, dan c, a ≠ b, b ≠ c, dan a ≠ c }
{ Harus dituliskan dari yang terkecil sampai dengan yang terbesar}

KAMUS
   a, b, c : integer

ALGORITMA
   input (a,b,c)
   depend on (a,b)
      a < b :  depend on (b,c)
                 b < c  : output (a,b,c)
                 b > c  : depend on (a,c)
                             c > a : output (a,c,b)
                             c < a : output (c,a,b)
      a > b :  depend on (a,c)
                 a < c  : output (b,a,c)
                 a > c  : depend on (b,c)
                             b < c : output (b,c,a)
                             b > c : output (c,b,a)
```

Contoh 4: RANKING (DENGAN PEMERIKSAAN KESALAHAN)

Persoalan :

Dibaca tiga buah harga a, b, dan c, harus dituliskan secara terurut, mulai dari yang terkecil sampai dengan yang terbesar. Ketiga bilangan yang dibaca selalu berlainan harganya.

Spesifikasi:

Input : a,b,c, tiga besaran integer
Proses : menuliskan harga yang dibaca mulai dari yang terkecil s.d. yang terbesar, jika a,b,c berlainan : output (a,b,c)
Output : Jika data benar ($a \neq b, b \neq c, a \neq c$):

a,b,c jika $a < b$ dan $b < c$
a,c,b jika $a < c$ dan $c < b$
b,a,c jika $b < a$ dan $a < c$
b,c,a jika $b < c$ dan $c < a$
c,a,b jika $c < a$ dan $a < b$
c,b,a jika $c < b$ dan $b < a$

Menuliskan pesan "Data salah" jika ada data yang sama

```
Program RANKING1
{ Dibaca tiga harga a, b, dan c }
{ harus dituliskan  dari yang terkecil sampai dengan yang terbesar}

KAMUS
   a, b, c : integer

ALGORITMA
   input (a,b,c)
   if (a ≠ b and b ≠ c and a ≠ c) then
     depend on (a,b,c)
        a < b < c   :  output (a,b,c)
        a < c < b   :  output (a,c,b)
        b < a < c   :  output (b,a,c)
        b < c < a   :  output (b,c,a)
        c < a < b   :  output (c,a,b)
        c < b < a   :  output (c,b,a)
   else { ada data yang sama : a=b or b=c or a=c }
     output ("Data salah, tidak sesuai spesifikasi")
```

Catatan:

1. Pemeriksaan data semacam itu akan menambah *robustness* dari program. Jika pemeriksaan data tidak dilakukan, maka kriteria data yang valid harus ditampilkan secara eksplisit sehingga pemakai program dapat mengetahui spesifikasi data benar dan tidak menyebabkan program *abort.*
2. Selanjutnya, algoritma-algoritma yang diberikan tidak akan mengandung pemeriksaan kesalahan.
3. Berikut ini diberikan skema pemrosesan data yang dibaca dengan pemeriksaan kesalahan yang dapat dipakai untuk beberapa kasus sederhana. Untuk kasus yang lebih kompleks, pemeriksaan tidak dapat dilakukan dengan cara sederhana dan pada awal program. Lihat latihan soal.

```
SKEMA PROSES_VALIDASI
{ Skema program yang menerima input data, hanya melakukan proses jika data
valid }

KAMUS
   { deklarasi data input }

ALGORITMA
   input (data)
   if (data_valid) then { data_valid adalah predikat, ekspresi boolean yang
                          menyatakan validitas dari data }
      { Lakukan proses }

   else {data tidak valid }
      output ("Data salah, tidak sesuai spesifikasi")
```

Contoh predikat yang merupakan ekspresi data valid:
- Jika data harus positif, dan data adalah X bertype integer: $X \geq 0$.
- Jika data adalah X dan Y, dan X harus lebih besar daripada Y: $X > Y$.

Latihan Soal

1. Data macam apa yang harus diberikan untuk "mencoba" semua contoh program analisis kasus di atas? Buatlah semua kemungkinan yang harus dicoba untuk melakukan test terhadap program-program contoh.

2. Bagaimana jika dalam persoalan RANKING ternyata ada bilangan yang sama ?

3. **Akar Persamaan Kuadrat**
   Dibaca 3 buah bilangan integer A, B, C yang mewakili koefisien persamaan kuadrat:
   $$AX^2 + BX + C$$
   Buatlah algoritma untuk menghitung akar-akar dari persamaan kuadrat tersebut. Tentukan data *test* untuk program anda.

4. **Bonus Pegawai**
   Tuliskanlah algoritma untuk menentukan bonus pegawai, berdasarkan ketentuan yang diberikan oleh bagian personalia dan keuangan sebagai berikut:
   Pegawai perusahaan digolongkan menjadi dua golongan, yaitu staf dan bukan staf. Staf akan mendapatkan bonus sebesar 1 juta rupiah dengan syarat bahwa ia telah bekerja paling tidak 5 tahun dan umurnya sudah mencapai 50 tahun; staf yang bekerja kurang dari 5 tahun berapapun umurnya, hanya mendapat bonus sebesar Rp. 500.000,00. Pegawai non-staf yang telah bekerja lebih dari 5 tahun akan mendapat bonus sebesar Rp. 400.000,00 jika berumur lebih dari 50 tahun sedangkan pegawai non staf yang berumur kurang dari 50 tahun hanya akan mendapat bonus Rp. 250.000,00. Pegawai staf yang umurnya kurang dari 50 tahun akan mendapat bonus Rp. 300.000,00.
   Apa komentar anda tentang ketentuan dari bagian personalia dan keuangan tersebut? Ingat bahwa anda seringkali harus membuat program berdasarkan "spesifikasi" semacam itu dari pemakai (*user*).

5. **SEGITIGA**
   Dibaca 3 buah bilangan riil sebagai data, yang mewakili panjang segmen garis dalam centimeter. Buatlah algoritma untuk menentukan apakah ketiga segment garis tersebut dapat membentuk sebuah segitiga. Output yang diharapkan adalah : "Dapat membentuk segitiga", jika ya atau "Tidak mungkin membentuk segitiga" jika tidak. Apa komentar Anda? Banyak persoalan yang harus diprogram, diformulasikan sejenis ini.

6. **TAHANAN**
   a. Dibaca tiga buah bilangan bulat yang mewakili tiga buah tahanan dalam Ohm: R1, R2, dan R3, dan sambungan yang akan dipilih "SERI" atau "PARALEL", harus dihitung dan dituliskan tahanan total yang dihasilkan sesuai dengan sambungan yang ditentukan.
   b. Bagaimana jika sambungan tersebut dikode? Jika dikehendaki untuk dilakukan pemeriksaan data, dan data yang valid untuk paralel berbeda dengan data valid untuk seri (sebab untuk tahanan paralel tidak diperbolehkan adanya data tahanan nol yang mengakibatkan pembagian dengan nol), maka skema PROSES&VALIDASI tidak dapat dipakai begitu saja, melainkan harus sedikit dimodifikasi. Tuliskanlah algoritmanya.
   c. Ubahlah deklarasi nama R1, R2, R3 menjadi ber-type bilangan riil. Apa dampaknya?

7. **GAJI**
   Pada suatu perusahaan, terdapat 5 golongan karyawan. Gaji karyawan ditentukan berdasarkan gaji tetap dan juga dari lamanya bekerja. Gaji tetap dan gaji per jam tersebut tergantung kepada golongan karyawan sesuai dengan tabel berikut

Golongan	**Gaji tetap**	**Gaji per jam**
1	Rp. 500.000,00	Rp. 5000,00
2	Rp. 300.000,00	Rp. 3000,00
3	Rp. 250.000,00	Rp. 2000,00
4	Rp. 100.000,00	Rp. 1500,00
5	Rp. 50.000,00	Rp. 1000,00

   Jika karyawan bekerja lebih dari 150 jam,kelebihan dari 150 jam tersebut dihitung sebagai lembur, dengan gaji per jam 1.5 kali gaji biasa.
   Dibaca data: golongan karyawan, nama karyawan, jam masuk/pulang kerja; harus dihitung gaji yang dibayar. Tentukan spesifikasi lembur dengan lebih persis.

8. Diberikan deretan analisis kasus sebagai berikut, untuk melakukan analisis terhadap nilai D (determinan) suatu persamaan kuadrat.
   Berikan komentar mengenai analisis kasus yang dituliskan dalam teks algoritma sebagai berikut:

```
Program Determinan
{ Dibaca a,b dan c yaitu nilai koefisien sebuah  persamaan kuadrat, harus
dituliskan "type" determinannya: negatif, nol atau positif }

KAMUS
    { Koefisien PK : }
     a : real
     b : real
     c : real

     D : real { determinan }

ALGORITMA
    input (a, b,c)
    D ← b*b - 4.0 * a * c
    if (D < 0) then
       output ("Determinan Negatif")
    if (D = 0) then
       output ("Determinan Nol")
    if (D > 0) then
       output ("Determinan Positif")
```

9. Berikan komentar mengenai analisis kasus yang dituliskan dalam teks algoritma sebagai berikut. Suatu ruangan mempunyai tigas buah lampu. Algoritma untuk menentukan lampu yang menyala dalam sebuah ruangan (dapat salah satu, dua atau ketiganya) tergantung sebuah nilai boolean yang mewakili lampu nyala/mati.

```
Program Lampu
{ Menentukan lampu yang menyala }

KAMUS
    { deklarasi status lampu }
    lampu1, lampu2, lampu3 : boolean { true jika harus menyala }

ALGORITMA
    { Potongan algoritma lain yang tidak ditulis }
    { State di titik ini : lampu1, lampu2 , lampu3 terdefinisi nilainya }
    if (lampu1) then
       output ("Nyalakan lampu1")
    if (lampu2) then
       output ("Nyalakan lampu2")
    if (lampu3) then
       output ("Nyalakan lampu3")
```

FUNGSI

Definisi

Fungsi adalah pemetaan suatu domain ke *range* berdomain tertentu. Fungsi adalah sebuah **transformasi akibat pemetaan suatu nilai (dari "domain") ke nilai lain (dalam "range")**. Secara algoritmik, sebuah fungsi akan menerima suatu harga yang diberikan lewat parameter formal bertype tertentu (jika ada) dan menghasilkan suatu nilai sesuai dengan domain yang didefinisikan dalam spesifikasi fungsi.

Dalam penulisannya, fungsi diberi **nama,** dan parameter formal yaitu harga masukan yang juga diberi nama dan dijelaskan typenya. Fungsi harus didefinisikan dalam kamus.

Fungsi yang didefinisikan dapat "dipanggil" untuk dieksekusi lewat namanya, dan dengan diberikan parameter aktualnya.

Penjelasan lebih rinci tentang parameter akan diberikan pada bagian Prosedur.

Contoh Fungsi:

Fungsi f(x) dengan satu parameter x dalam matematika yang didefinisikan sebagai:

$$f(x) = x^2 + 3x - 5$$

jika x = 4 maka f(x) akan menghasilkan 23
jika x = 1 maka f(x) akan menghasilkan -1

$f(x,y) = x^2 + 3xy - 5y - 1$ adalah fungsi dengan dua parameter x dan y
jika diberi harga x = 0 dan y = 0 maka f(x,y) akan menghasilkan -1
jika diberi harga x = 1 dan y=0 maka f(x,y) akan menghasilkan 0

Notasi Algoritmik untuk Fungsi

Pendefinisian/Spesifikasi Fungsi

```
function NAMAF (<list-parameter-input>) → <type-hasil>
{ Spesifikasi fungsi: diberikan ... menghasilkan ... }
```

```
KAMUS LOKAL
{ Semua nama yang dipakai dalam algoritma/realisasi fungsi }
```

```
ALGORITMA
{ Deretan instruksi algoritmik :
  pemberian harga, input, output, analisis kasus, pengulangan }

{ Pengiriman harga di akhir fungsi, harus sesuai dengan type hasil}
   → <hasil>
```

dengan syarat :

- <list-parameter-input> boleh tidak ada (kosong), dalam hal ini di fungsi tidak membutuhkan apa-apa dari pemakainya untuk menghasilkan harga.
- Jika <list-parameter-input> (list parameter formal) tidak kosong, minimal mengandung satu nama, maka nama tersebut harus berupa nama informasi beserta type-nya.
- Instruksi "terakhir" yang harus ada pada fungsi harus merupakan pengiriman harga yang dihasilkan oleh fungsi (dituliskan seperti pada notasi di atas, dengan <type-hasil> boleh type dasar atau type terstruktur). <type-hasil> boleh dinyatakan oleh suatu nama type. Dengan catatan, bahwa instruksi "terakhir" belum tentu dituliskan pada baris terakhir, misalnya jika hasil merupakan sebuah nilai yang dikirimkan berdasarkan analisis kasus.

Pemanggilan Fungsi

```
Program POKOKPERSOALAN
{Spesifikasi : Input, Proses, Output}

KAMUS
{ Semua nama yang dipakai dalam algoritma }

   function NAMAF (<list-nama-parameter-formal/input>) → <type-hasil>
   { Spesifikasi fungsi }

ALGORITMA
{ Deretan instruksi pemberian harga, input, output, analisis kasus,
  pengulangan yg memakai fungsi }
{ Harga yang dihasilkan fungsi juga dapat dipakai dalam ekspresi }

   nama ← NAMAF(<list-parameter-aktual>)

   output ( NAMAF(<list-parameter-aktual>) )

{ Harga yang dihasilkan fungsi juga dapat dipakai dalam ekspresi }
```

Catatan :

1. Pada waktu pemanggilan terjadilah asosiasi antara parameter formal/input dengan parameter aktual sesuai dengan urutan penulisan dalam <list-nama-parameter-formal/input>.
2. Parameter input dapat berupa nama informasi atau nama konstanta yang telah terdefinisi dalam kamus atau konstanta; dapat juga berupa harga konstanta, atau harga yang dihasilkan oleh suatu ekspresi atau fungsi.
3. <list-parameter-aktual> harus sama jumlah, urutan dan type-nya dengan <list-parameter-formal/input> pada pendefinisian fungsinya.
4. Harga yang dihasilkan oleh fungsi dapat didefinisikan domainnya dengan lebih rinci.
5. Pada akhir dari eksekusi fungsi, harga yang dihasilkan oleh fungsi dikirimkan ke pemakainya.
6. Fungsi boleh dipakai oleh program utama, prosedur atau fungsi lain.

Fungsi Terdefinisi

Fungsi terdefinisi adalah fungsi yang sudah diberikan oleh sistem dan tinggal dipakai (dipanggil). Fungsi terdefinisi selalu diberikan daftar dan spesifikasinya (domain + *range*).

Fungsi terdefinisi untuk Melakukan **Konversi Tipe**:
Seringkali, dibutuhkan konversi dari bilangan riil menjadi integer atau sebaliknya. Maka didefinisikan duabuah fungsi konversi bilangan numerik sebagai berikut:

```
function RealToInteger (x : real) → integer
{ Mengkonversi harga x yang bertype real menjadi harga lain yang betype
integer dengan pemotongan, yaitu menghilangkan bagian di belakang titik
desimalnya}

function IntToReal (i : integer) → real
{ Mengkonversi harga i yang bertype integer menjadi harga ekivalen yang
betype real }
```

Fungsi-fungsi terdefinisi untuk **seleksi** terhadap sebuah string:

```
function AWAL (S : string) → string
{ Menghasilkan sisa string S tanpa karakter terakhir }

function AKHIR (S : string) → string
{ Menghasilkan sisa string S tanpa karakter pertama }

function FIRSTCHAR (S : string { tidak kosong } ) → character
{ Menghasilkan karakter pertama string S }

function LASTCHAR (S : string { tidak kosong } ) → character
{ Menghasilkan karakter terakhir string S }
```

Fungsi-fungsi untuk **memperoleh informasi** tentang sebuah string:

```
function LONG (S : string) → integer
{ Menghasilkan panjang string S}

function IsKOSONG (S : string) → boolean
{Menghasilkan true jika S adalah string kosong }
```

Fungsi **suksesor** dan **predesesor** untuk integer:

```
function Succ (x : integer) → integer
{ Menghasilkan suksesor x, x+1 }
{ Contoh: Succ(0) = 1
          Succ(-1) = 0
          Succ(5) = 6   }

function Pred (x : integer) → integer
{ Menghasilkan predesesor x, x - 1 }
{ Contoh: Pred(-1) = -2
          Pred(3) = 2    }
```

Fungsi **matematik**:

```
function Sin (x : real) →  real
{ Menghasilkan sinus(x), x dalam radian }

function Cos (x : real) →real
{ Menghasilkan cosinus(x), x dalam radian }

function Abs (x : integer) → integer
{ Menghasilkan  |x| }
```

Contoh Penulisan Algoritmik

Berikut ini diberikan contoh pendefinisian dan pemakaian (pemanggilan) fungsi:

Pendefinisian Fungsi

```
function FX_KUADRAT (x : integer) → integer
{ Diberikan x, integer, menghitung f(x) = x^2 + 3x - 5 }
KAMUS LOKAL
ALGORITMA
   → (x * x + 3 * x - 5)
```

```
function FXY (x, y : real) → real
{ Diberikan x dan y, real, menghitung f(x) = x^2 + 3xy - 5y - 1 }
KAMUS LOKAL
ALGORITMA
   → (x * x + 3 * x * y - 5 * y - 1)
```

Pemanggilan Fungsi

```
Program CONTOHF1
{ Dibaca x dan y, menghitung :  }
{ f(x,y) = x^2 + 3xy - 5y - 1; f(x) = x^2 + 3x - 5  }
{ dan menuliskan hasil perhitungan }
```

```
KAMUS
   x : integer  { data }
   y : real     { data }
   FX : integer { Hasil perhitungan f(x) = x^2 + 3x - 5 }
   FY : real    { Hasil perhitungan f(x,y) = x^2 + 3xy - 5y - 1 }

   function FXY (x, y : real) → real
   { Diberikan x dan y, menghitung f(x,y) = x^2 + 3xy - 5y - 1 }

   function FX_KUADRAT (x : integer) → integer
   { Diberikan x, menghitung f(x) = x^2 + 3x - 5 }
```

```
ALGORITMA
   input (x,y)
   FX ← FX_KUADRAT(x)
   FY ← FXY(IntToReal(x),y)
   output (FX,FY)
```

```
Program CONTOHF2
{ Dibaca x dan y, menghitung : }
{ f(x) = x^2 + 3x - 5;    f(x,y) = x^2 + 3xy - 5y - 1 }
{ dan menuliskan hasil perhitungan }
```

```
KAMUS
   x, y : integer  { data }

   function FX_KUADRAT (x : integer) → integer
   { Diberikan x, menghitung f(x) = x^2 + 3x - 5 }
   function FXY (x, y : real) → real
   { Diberikan x dan y, menghitung f(x,y) = x^2 + 3xy - 5y - 1 }
```

```
ALGORITMA
   input (x,y)
   output (FX_KUADRAT(x), FXY(IntToReal(x), IntToReal(y))
```

Contoh 1: FUNGSI KONVERSI

Persoalan:

Tuliskanlah sebuah fungsi, yang mengkonversikan harga karakter angka (nol sampai dengan 9) menjadi harga numerik sesuai dengan karakter yang tertulis.

Contoh : '0' → 0
'8' → 8

Spesifikasi:

Fungsi KarakterToInteger:
Domain : x : character ['0'..'9'])
Range : integer [0..9]
Proses : Analisis kasus terhadap x, untuk setiap harga x diasosiasikan integer yang sesuai.

```
function KarakterToInteger (x : character['0'..'9']) → integer[0..9]
{ Diberikan x berupa karakter, menghasilkan harga integer yang sesuai dengan
penulisan pada karakter  }
```

```
KAMUS LOKAL
```

```
ALGORITMA
   depend on  (x)
      x = '0' : → 0
      x = '1' : → 1
      x = '2' : → 2
      x = '3' : → 3
      x = '4' : → 4
      x = '5' : → 5
      x = '6' : → 6
      x = '7' : → 7
      x = '8' : → 8
      x = '9' : → 9
```

Contoh 2: APAKAH HURUF 'A'
Pemetaan karakter ke type boolean

Persoalan :

Tuliskanlah fungsi IsAnA yang mentest apakah sebuah karakter yang diberikan kepadanya adalah sebuah huruf 'A'. Harga yang dihasilkan adalah benar jika huruf itu 'A', salah jika huruf itu bukan 'A'.

Contoh : IsAnA('A') → true
IsAnA ('X') → false
IsAnA ('Y') → false

Spesifikasi :

Fungsi IsAnA :
Domain : x (karakter)
Range : boolean
Proses : menghasilkan true jika x adalah 'A', false jika tidak

```
function IsAnA (x : character) → boolean
{ Menghasilkan true jika x adalah 'A'; dengan menuliskan ekspresi booelan }
KAMUS LOKAL
ALGORITMA
   → (x = 'A')
```

```
function IsAnA1 (x : character) →  boolean
{ Mengetes apakah x adalah 'A': true jika x adalah 'A'; dengan melakukan
analisis kasus terhadap x }
KAMUS LOKAL
ALGORITMA
   if  (x = 'A') then
      →  true
   else { x ≠ 'A' }
      → false
```

Contoh 3: HITUNG METER+CM

Pemetaan type dasar ke type bentukan

Persoalan:

Tuliskanlah sebuah fungsi, yang jika diberikan sebuah angka Cm yang menyatakan panjang dalam cm, akan menghasilkan pasangan harga <x1, x2> sesuai dengan rumus ukuran metris (i m = 100 cm).sehingga x1*100+x2 = Cm

Contoh : F(100) = <1,0>
F(355) = < 3, 55>

Spesifikasi :

Fungsi KonversiCm :
Domain : Cm : integer
Range : pasangan harga integer
Proses : menghitung Meter dan SisaCm sehingga Cm = 100* Meter + SisaCm

```
function KonversiCm1 (Cm : integer) → < integer, integer >
{ Diberikan Cm, mengubahnya menjadi berapa meter dan cm }
KAMUS LOKAL
   meter  : integer   { meter }
   sisaCm : integer   { sisa cm }
ALGORITMA
   meter ← Cm div 100
   sisaCm ← Cm mod 100
   →  < meter, sisaCm >
```

```
function KonversiCm2 (Cm : integer) → < integer, integer >
{ Diberikan Cm, mengubahnya menjadi berapa meter dan cm }
```

```
KAMUS LOKAL
```

```
ALGORITMA
   →  < Cm div 100, Cm mod 100 >
```

Catatan:

1. Terjemahan teks fungsi pada contoh ini ke dalam beberapa bahasa pemrograman tidak "sederhana", bahkan tidak mungkin dapat dilakukan secara langsung.
2. Cobalah menterjemahkan fungsi di atas ke dalam bahasa Pascal, C dan Ada. Akan terlihat perbedaan yang sangat besar di antara ketiganya.

Contoh 4: UBAHDANPERIKSAKAR

Persoalan:

Tuliskanlah algoritma yang membaca sebuah karakter dan mengubahnya menjadi integer yang sesuai dengan karakter itu jika karakter yang dibaca adalah antara '0' dan '9' dengan memanfaatkan funsgi KarakterToInteger yang pernah dibuat. . Jika bukan di dalam daerah harga nol dan sembilan, maka harus dituliskan pesan yang berbunyi : 'Bukan angka' .

Spesifikasi:

Input : CC karakter
Output : menuliskan integer [0..9] sesuai dengan karakter yang dibaca, jika karakter ['0'..'9'], atau "Bukan angka" jika CC bukan angka
Proses : Konversi dari karakter ke integer, jika CC ['0'..'9']
Dengan catatan fungsi konversi adalah seperti terdefinisi pada contoh 1.

Catatan:

Tes yang dilakukan sebelum pemanggilan fungsi KarakterToInteger membuat fungsi KarakterToInteger cukup mengembalikan nilai yang terdefinisi. Jika domain dari nilai masukan fungsi tidak hanya mencakup karakter angka, maka di dalam fungsi harus didefinisikan suatu nilai yang diluar definisi angka.

```
Program UbahdanPeriksaKar
{ Program membaca sebuah karakter, dan melakukan konversi ke nilai integer
serta menuliskannya, jika karakter mewakili angka, bernilai ['0'..'9'.
Pogram menulis "bukan angka", jika yang diketik bukan bernilai ['0'..'9'] }
```

```
KAMUS
   CC :  character    {data, karakter yang dibaca}

   function KarakterToInteger (x : character['0'..'9']) → integer[0..9]
   { Diberikan x berupa karakter '0'..'9', menghasilkan harga integer
     yang sesuai dengan penulisan pada  karakter }
```

```
ALGORITMA
   input (cc)
   depend on (CC)
      CC ∈ ['0'..'9'] : output (KarakterToInteger(CC))
      CC ∉ ['0'..'9'] : output ("Bukan angka")
```

Contoh 5: HITUNGFUNGSI
Definisi dan pemanggilan fungsi

Persoalan:

Tuliskanlah algoritma yang membaca 3 bilangan bulat (a,b,c), dan menghitung:
$6*(ax^2 + bx +c)$ untuk x = 1

Spesifikasi:

Input : a, b, c bilangan bulat, koefisien persamaan kuadrat, x = 1
Output : Fx
Proses : menghitung $Fx = 6*(ax^2 + bx +c)$
menuliskan hasil

```
Program HITUNGFUNGSI
{ Program membaca tiga buah bilangan bulat a,b,c dan menghitung nilai fungsi
F(x) = 6*(ax^2 + bx +c) }
```

```
KAMUS
   a, b, c : integer   { data, koefisien persamaan kuadrat }
   Fx      : integer   { hasil perhitungan persamaan }

   function FungsiKUADRAT (a, b, c, x : integer) → integer
   { Diberikan a, b, c, x menghitung f(x) = ax^2 + bx +c }
```

```
ALGORITMA
   input (a,b,c)
   Fx ←.6 * FungsiKUADRAT (a,b,c,1)
   output (Fx)
```

```
function FungsiKUADRAT (a, b, c, x : integer) → integer
{ Diberikan a, b, c, x menghitung f(x) = ax^2 + bx +c }
```

```
KAMUS LOKAL
```

```
ALGORITMA
      → (a * x * x + b * x +c)
```

Contoh 6: MAX2 dan MAX3

a. Tuliskanlah fungsi MAX2, yang menerima masukan dua buah bilangan real dan menghasilkan sebuah bilangan real, yaitu salah satu di antara nilai dua buah bilangan tersebut yang terbesar.
b. Kemudian **dengan memakai fungsi MAX2**, tuliskanlah sebuah fungsi lain MAX3 yang menghasilkan nilai terbesar dari tiga buah bilangan real.
   Contoh : MAX2(1,2)) → 2
   MAX2(10,2) →10
   MAX3(1,2,3) adalah MAX2(MAX2(1,2),3) → 3
   MAX3(10,2,3) adalah MAX2(MAX2(10,2),3) → 10 }

```
function MAX2 (a, b : real) →real
{ Diberikan a dan b, menghasilkan a jika a ≥ b,
  menghasilkan b jika b > a }
KAMUS LOKAL
ALGORITMA
   depend on (a,b)
      a ≥ b : →a
      b > a : →b
```

```
function MAX3 (a, b, c : real) → real
{ Diberikan a, b, dan c, menghasilkan a jika a ≤ b dan a ≤ c, menghasilkan b
jika b ≤ a dan b ≤ c, menghasilkan c jika c ≤ b dan c  ≤ a }
KAMUS LOKAL
ALGORITMA
   →MAX2 (MAX2(a,b),c)
```

```
function  MAX3BIS (a, b, c : real) → real
{ Diberikan a, b, dan c, menghasilkan a jika a ≤ b dan a ≤ c, menghasilkan b
jika b ≤ a dan b ≤ c, menghasilkan c jika c ≤ b dan c ≤ a }
KAMUS LOKAL
ALGORITMA
      →MAX2 (a,MAX2(b,c))
```

Catatan :

1. Bagi yang sudah mengikuti kuliah "Pemrograman Fungsional", dapat dilihat bahwa teks program di atas sangat selaras dengan ekspresi fungsional yang pernah dipelajari.
2. Pemanggilan fungsi pada konteks prosedural adalah aplikasi fungsi pada konteks fungsional.

Contoh-7: PENANGGALAN DAN NEXTDAY

Pemetaan type bentukan ke type bentukan

Didefinisikan type terstruktur untuk mewakili hari seperti dalam kalender yang kita pakai sehari-hari:

```
type  DATE : < DD : Tanggal, MM : Bulan, YY : Tahun >
```

Konstanta :
```
<5,12,1990>       { artinya 5 Desember 1990 }
<25,2,2001>       { artinya 25 Februari 2001 }
```

Tuliskanlah algoritma untuk :

- Membaca sebuah tanggal dan sebuah kode bahasa penulisan (1 = Inggris, 2 = Indonesia, 3 = Perancis),

- Menuliskan tanggal keesokan harinya sesuai dengan kode bahasa:
  Dalam bahasa Inggris , DATE ditulis dalam: Bulan, '/',Tanggal.'/',Tahun
  Dalam bahasa Indonesia, DATE ditulis dalam: Tanggal, '-',Bulan.'-',Tahun
  Dalam bahasa Perancis DATE ditulis dalam: Tanggal, '/',Bulan.'/',Tahun
- Proses menghitung hari esok dilakukan oleh sebuah fungsi NextDay yang menerima masukan sebuah tanggal dan menghasilkan tanggal keesokan harinya. Contoh pemanggilan dan hasil fungsi diberikan sebagai berikut:
  NextDay(<13,4,1990>,1) → 4 /14/1990
  NextDay(<30,1,1990>,2) → 31-1-1990
  NextDay(<31,12,1990>,3) → 1/1/1991

```
Program PENANGGALAN

KAMUS
   type Tanggal : integer [1..31]
   type Bulan   : integer [1..12]
   type Tahun   : integer > 0
   type DATE    : < DD : Tanggal, MM : Bulan, YY : Tahun >

   HariIni, Esok : DATE
   KodeBahasa : integer [1..3]

   function NEXTDAY (Now : DATE) → DATE
   { Mengirimkan keesokan hari dari Now }

ALGORITMA
   input (HariIni, KodeBahasa)  { membaca <Tanggal,Bulan,Tahun> dan
                                  Kodebahasa }
   Esok ← NEXTDAY(HariIni)
   depend on KodeBahasa
      KodeBahasa = 1 : output (Esok.MM, "/", Esok.DD, "/", Esok.YY)
      KodeBahasa = 2 : output (Esok.DD, "-", Esok.MM, "-", Esok.YY)
      KodeBahasa = 3 : output (Esok.DD, "/", Esok.MM, "/", Esok.YY)
```

```
function NEXTDAY (Now : DATE) → DATE
{ Menerima Now dan menghasilkan tanggal keesokan hari dari Now }

KAMUS LOKAL
    function Kabisat (yy : integer) →  boolean
    { true jika yy adalah tahun kabisat }

ALGORITMA
   depend on (Now.MM)
      Now.MM = 2 : { bulan Februari, kasus kabisat dan bukan }
         depend on Now.DD
            Now.DD < 28 : → <Now.DD+1,Now.MM,Now.YY>
            Now.DD = 28 :
               depend on Now.YY
                 not Kabisat(Now.YY) : → <1,Now.MM+1,Now.YY>
                 Kabisat(Now.YY)     : → <Now.DD+1,Now.MM,Now.YY>
            Now.DD = 29 : → <1,Now.MM+1,Now.YY>
      Now.MM = 12 : { mungkin pergantian tahun }
         depend on Now.DD
            Now.DD < 31 : → <Now.DD+1,Now.MM,Now.YY>
            Now.DD = 31 : → <1,1,Now.YY + 1>
      Now.Bulan ∈ [1,3,5,7,8,10] : { sebulan ada 31 hari }
         depend on Now.DD
            Now.DD < 31 : → <Now.DD+1,Now.MM,Now.YY>
            Now.DD = 31 : → <1,Now.MM+1,Now.YY>
      Now.Bulan ∈ [4,6,9,11] : { sebulan ada 30 hari }
         depend on Now.DD
            Now.DD < 30 : → <Now.DD+1,Now.MM,Now.YY>
            Now.DD = 30 : → <1,Now.MM+1,Now.YY>
```

Soal untuk algoritma di atas:

- Tuliskanlah definisi fungsi KABISAT dan algoritmanya.
- Buat analisis kasus yang lain dari yang dituliskan pada fungsi NEXTDAY di atas, dengan mulai menganalisis tanggal dan bukan bulan.
- Buatlah kode program dari fungsi Nextday yang sama dengan hanya melakukan komputasi aritmatika, seperti halnya pada perhitungan Durasi yang dibahas pada kalkulasi Type terstruktur JAM (pada penjelasan *sequence*).

Latihan Soal

1. Domain dan *range* dari fungsi secara umum dapat dirumuskan sebagai berikut:

DOMAIN	RANGE
type dasar	type dasar
type dasar	type bentukan
type bentukan	type dasar
type bentukan	type bentukan

   Definisikan untuk masing-masing kategori tersebut tiga buah fungsi. Tuliskanlah algoritmanya.

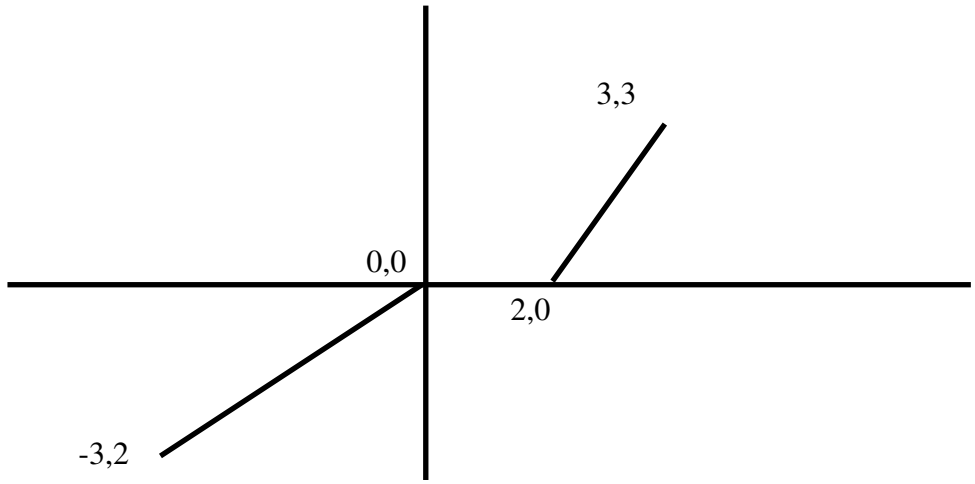
2. Pada beberapa kasus, dikehendaki bahwa fungsi merupakan parameter. Sangat praktis jika nama fungsi dapat dipakai sebagai parameter. Pelajarilah cara

implementasi nama fungsi sebagai parameter dalam beberapa bahasa pemrograman.

3. Pada beberapa bahasa (Fortran, Pascal) fungsi hanya dapat menghasilkan harga ber-type dasar. Bahkan pada FORTRAN hanya dapat menghasilkan type numerik (integer, real). Bagaimana cara merealisasi fungsi yang menghasilkan type bentukan?

4. Hitung Fungsi Linier dan Patah-Patah

Tuliskanlah sebuah fungsi bernama FLINIER yang menerima input sebuah bilangan real dan menghitung harga f(x) sesuai dengan kurva yang melewati titik-titik sebagai berikut:



5. Pecahan

1. Definisikan sebuah type pecahan yang terdiri dari pembilang dan penyebut bilangan integer, dan sekumpulan fungsi yang merupakan realisasi dari operator pecahan:
    a. JumlahP : menerima dua buah pecahan, menghasilkan jumlah berupa pecahan.
    b. KurangP : menerima dua buah pecahan, menghasilkan selisih berupa pecahan.
    c. KaliP : menerima dua buah pecahan, menghasilkan hasil kali berupa pecahan.
    d. BagiP : menerima dua buah pecahan, menghasilkan hasil bagi berupa pecahan.
2. Buat soal di atas jika pecahan terdiri dari bilangan bulat, pembilang dan penyebut, dengan syarat bahwa pembilang harus lebih kecil dari penyebut.
3. Bagi yang sudah mengikuti kuliah pemrograman fungsional dengan menggunakan Diktat Pemrograman Fungsional oleh penulis yang sama, lihat pembahasan mengenai Type PECAHAN, kemudian buatlah semua operator menjadi fungsi dalam notasi prosedural. Implementasikan dalam sebuah unit/teks terpisah yang akan mengelola semua operasi terhadap pecahan dengan kerangka sebagai berikut:

```
MODUL PECAHAN
{ Berisi definisi dan semua primitif pemrosesan Pecahan }

KAMUS UMUM
{ Definisi type Pecahan }
   ..............
{ Spesifikasi semua fungsi yang berhubungan dengan Pecahan }
   ..............
{ Memeriksa apakah dua buah nilai integer dapat membentuk sebuah Pecahan
valid }
   ..............
{ Operator aritmetika Pecahan : Penjumlahan, pengurangan, pembagian,
                                perkalian, ... }
   ..............
{ Operator relasional Pecahan  : >, <, =, ≠ }
   ..............

ALGORITMA
{ Realisasi dari setiap fungsi dalam kamus tersebut }
```

6. TYPE JAM dan DATE (lihat bagian Type)

Bagi yang sudah mengikuti kuliah pemrograman fungsional, dengan berinspirasi kepada type Pecahan, buatlah semua operator menjadi fungsi dalam notasi prosedural. Implementasikan dalam sebuah unit/teks terpisah yang akan mengelola semua operasi terhadap pecahan dengan kerangka sebagai berikut:

```
MODUL TYPE XXX
{ Berisi definisi dan semua primitif pemrosesan TYPE XXX }

KAMUS UMUM
{ Definisi type }
   ..............
{ Daftar FUNGSI PRIMITIF }
   ..............
{ Memeriksa validitas nilai komponen untuk dibentuk menjadi type ybs. }
   ..............
{ Operator aritmetika Jam : Penjumlahan, pengurangan, pembagian,
                            perkalian, ... }
   ..............
{ Operator relasional Jam : >, <, =, ≠ }
   ..............

ALGORITMA
{ Realisasi dari fungsi dalam kamus tersebut }
```

PROSEDUR

Definisi

Prosedur adalah sederetan instruksi algoritmik yang diberi nama, dan akan menghasilkan efek neto yang terdefinisi. Prosedur menyatakan suatu aksi dalam konsep algoritma yang dibicarakan pada cerita "Mengupas kentang".

Mendefinisikan (membuat spesifikasi) prosedur berarti menentukan nama prosedur serta parameternya (jika ada), dan mendefinisikan **keadaan awal (*Initial State*, I.S.)** dan **keadaan akhir (*Final State*, F.S.)** dari prosedur tersebut. Prosedur didefinisikan (dituliskan spesifikasinya) dalam **kamus**. Cara penulisan spesifikasi : prosedur diberi **nama,** dan **parameter formal** (jika ada) yang juga diberi nama dan dijelaskan typenya.

Secara sederhana, dapat diartikan bahwa sebuah prosedur yang terdefinisi "disimpan" di tempat lain, dan ketika "dipanggil" dengan menyebutkan namanya "seakan-akan" teks yang tersimpan di tempat lain itu menggantikan teks pemanggilan. Pada saat itu terjadi asosiasi parameter (jika ada). Dengan konsep ini, maka I.S dan F.S dari prosedurlah yang menjamin bahwa eksekusi program akan menghasilkan efek neto yang diharapkan.

Jadi, setiap prosedur harus:
- didefinisikan (dibuat spesifikasinya) dan dituliskan kode programnya,
- dipanggil, pada saat eksekusi oleh prosedur lain atau oleh program utama.

Parameter Prosedur

Prosedur tanpa parameter memanfaatkan nilai dari nama-nama yang terdefinisi pada kamus global. Pemakaiannya biasanya harus "hati-hati", apalagi jika teks program sudah sangat besar dan implementasinya menjadi banyak file.

Prosedur berparameter dirancang, agar sepotong kode yang sama ketika eksekusi dilakukan, dapat dipakai untuk nama parameter yang bebeda-beda.

Nama parameter yang dituliskan pada definisi/spesifikasi prosedur disebut sebagai parameter formal. Sedangkan parameter yang dituliskan pada pemanggilan prosedur disebut sebagai parameter aktual.

Parameter formal adalah nama-nama variabel (list nama) yang dipakai dalam mendefinisikan prosedur, dan membuat prosedur tersebut dapat dieksekusi dengan nama-nama yang berbeda ketika dipanggil. Parameter formal adalah list nama yang akan dipakai pada prosedur, yang nantinya akan diasosiasikan terhadap nama variabel lain pada saat pemanggilan. Sesuai dengan ketentuan nilainya, ada tiga type parameter formal:

- parameter input, yaitu parameter yang diperlukan prosedur sebagai masukan untuk melakukan aksi yang efektif.
- parameter output, yaitu parameter yang nilainya **akan** dihasilkan oleh prosedur. Hasil nilai akan disimpan pada nama parameter Output ini.
- parameter input/output, yaitu parameter yang nilainya diperlukan prosedur sebagai masukan untuk melakukan aksi, dan pada akhir prosedur akan dihasilkan nilai yang baru.

Pemanggilan Prosedur

Memakai, atau "memanggil" prosedur adalah menuliskan nama prosedur yang pernah didefinisikan, dan memberikan harga-harga yang dibutuhkan oleh prosedur itu untuk dapat melaksanakan suatu aksi terdefinisi. Sebuah prosedur juga boleh "memakai" atau memanggil prosedur. Pada saat pemanggilan terjadi “passing parameter”.

Parameter aktual adalah nama-nama informasi yang dipakai ketika prosedur itu dipakai ("dipanggil"). Parameter aktual dapat berupa nama atau harga, tetapi harus berupa nama jika parameter tersebut adalah parameter output (karena hasilnya akan disimpan dalam nama tersebut). Sesuai dengan jenis parameter formal, parameter aktual pada saat pemanggilan:
- parameter **input** harus terdefinisi nilainya (karena dibutuhkan oleh prosedur untuk menghasilkan nilai). Karena yang dibutuhkan untuk eksekusi hanya nilai, maka parameter input dapat digantikan dengan suatu nilai tanpa menggunakan nama.
- parameter **output** tidak perlu terdefinisi nilainya, tetapi justru setelah pemanggilan prosedur akan dimanfaatkan oleh deretan instruksi berikutnya, karena nilainya akan dihasilkan oleh prosedur. Karena parameter output menampung hasil, maka harus berupa nama, dan tidak boleh diberikan nilai saja.
- parameter **input/output** harus terdefinisi nilainya dan nilai baru yang diperoleh karena eksekusi prosedur akan dimanfaatkan oleh deretan instruksi berikutnya. Seperti halnya parameter output, maka paramteter aktual harus berupa nama.

Pada saat eksekusi, terjadi asosiasi nama parameter formal dengan nama parameter aktual. Pada notasi algoritmik, asosiasi dilakukan dengan cara “by position”. urutan nama pada parameter aktual akan diasosiasikan sesuai dengan urutan parameter formal. Karena itu, type harus kompatibel.

Beberapa bahasa pemrograman, dapat dilakukan asosiasi dengan nama dan memperbolehkan adanya *nilai default*. Beberapa bahasa pemrograman (Ada, C) juga memperbolehkan nama prosedur yang sama, tetapi parameternya berbeda (*overloading*).

Prosedur dapat mempunyai **kamus lokal**, yaitu pendefinisan nama yang dipakai dan hanya berlaku dalam ruang lingkup prosedur tersebut. Jika nama yang dipakai di dalam prosedur tidak terdefinisi dalam list parameter formal atau dalam kamus lokal, maka nama tersebut harus sudah terdefinisi pada prosedur yang memakainya. Penulisan kamus lokal sama dengan kamus global, yang berbeda adalah lingkup berlakunya nama yang didefinisikan:
- pada kamus "global", nama berlaku untuk program dan semua prosedur/fungsi yang didefinisikan.

- pada kamus lokal, nama berlaku untuk prosedur/fungsi yang bersangkutan dan prosedur / fungsi yang didefinisikan di dalamnya.
- nilai yang disimpan dalam nama yang didefinisikan pada kamus lokal, hanya akan terdefinisi selama eksekusi prosedur, dan tidak dikenal lagi oleh pemanggilnya.

Program yang moduler adalah program yang dibagi-bagi menjadi modul-modul yang terdefinisi dengan baik dalam bentuk prosedur-prosedur. Setiap prosedur harus jelas definisi dan ruang lingkupnya, supaya dapat dipanggil secara independent. Pembagian program besar dalam prosedur-prosedur akan mempermudah pembagian kerja di antara beberapa pemrogram. Penulisan prosedur juga akan memudahkan program untuk dibaca oleh "manusia" karena kita tidak perlu terpaku pada detil kode prosedur untuk mengerti efek neto yang dihasilkannya. Bahkan dalam beberapa hal, pemrogram tidak perlu tahu sama sekali "isi" atau kode dari prosedur dengan mengetahui spesifikasinya, beberapa bahasa pemrograman bahkan menyediakan prosedur terdefinisi yang sering dipakai dalam memrogram sehingga pemrogram tidak perlu lagi menuliskan kodenya.

Prosedur berlaku untuk ruang lingkup (*universe*) tertentu, terutama untuk prosedur yang tidak mempunyai parameter. Dalam dua bab berikut, yaitu Mesin Gambar dan Mesin Karakter, akan diberikan gambaran lebih jelas dan lengkap tentang pendefinisian dan pemakaian prosedur karena keduanya adalah mesin abstrak yang tertentu.

Notasi Algoritmik untuk Prosedur

Pada bagian ini diberikan skema pendefinisian dan pemanggilan prosedur.

Pendefinisian/Spesifikasi Prosedur

```
procedure NAMAPROSEDUR ( [<list-parameter-formal>])
{ Spesifikasi, Initial State, Final State }
```

```
KAMUS LOKAL
{ Semua nama yang dipakai dalam BADAN PROSEDUR }
```

```
ALGORITMA
{ BADAN PROSEDUR }
{ Deretan instruksi pemberian harga, input, output, analisis kasus,
  Pengulangan, atau prosedur }
```

dengan syarat :

- Nama prosedur dan prameternya harus disebutkan dalam kamus pemanggil.
- `<list-parameter-formal>` boleh tidak ada (kosong), dalam hal ini di dalam prosedur akan dipakai nama lokal dan nama-nama yang telah terdefinisi dalam kamus ”pemakai”-nya.
- Jika `<list-parameter-formal>` ada (tidak kosong, minimal satu nama), maka harus berupa satu atau beberapa nama INFORMASI beserta type-nya.

Pemanggilan Prosedur

```
Program POKOKPERSOALAN
{ Spesifikasi, Input, Proses, Output }

KAMUS
{ Semua nama yang dipakai dalam algoritma }

   procedure NAMAPROSEDUR (<list-parameter-formal>)
   { Spesifikasi : Initial State, Final State }

ALGORITMA
{ Deretan instruksi assignment/pemberian harga, input, output, analisis
  kasus, pengulangan )

   NAMAPROSEDUR  (<list-parameter-aktual>)
```

dengan syarat :

- Pada waktu pemanggilan terjadilah asosiasi antara parameter formal dengan parameter aktual sesuai dengan urutan penulisan dalam `<list-parameter-formal>`.
- `<list-parameter-aktual>` harus sama jumlah, urutan dan typenya dengan `<list-parameter-formal>`.
- `<list-parameter-aktual>` yang berupa **input** dapat berupa **nama informasi** atau nama **konstanta/ekspresi** yang telah terdefinisi dalam kamus atau konstanta; dapat juga berupa harga konstanta, atau harga yang dihasilkan oleh suatu ekspresi atau fungsi.
- `<list-parameter-aktual>` yang berupa **input/output** atau **output harus** berupa **nama informasi.** Jika didefinisikan sebagai **Input**, walaupun pernah diubah dalam badan prosedur, isi dari nama yang dipakai pada parameter aktual tidak pernah berubah. Jika didefinisikan sebagai parameter **Output** dan parameter aktual yang diasosiasikan terhadapnya pernah diubah harganya dalam badan prosedur, isinya akan berubah.

Contoh

Contoh 1: VOLTAGE

Tuliskanlah program yang membaca tahanan (Ohm) dan arus (Ampere), kemudian menghitung tegangan yang dihasilkan dan menuliskan hasilnya. Perhitungan tegangan harus dituliskan menjadi suatu prosedur bernama PROSES, supaya struktur program jelas : Input - Proses - Output.

Input : R : integer, tahanan (Ohm) dan A : integer, arus (Ampere)
Proses : menghitung V= R * A
Output : V : integer, tegangan (Volt)

Pelajarilah dua buah solusi yang diberikan berikut ini, dan berikan komentar anda.

Solusi 1: Prosedur tanpa parameter

```
Program VOLTAGE1
{ Program yang membaca tahanan dan arus, menghitung Voltage dan mencetak
hasil perhitungan }

KAMUS
   R : integer     { tahanan dalam ohm }
   A : integer     { arus dalam ampere }
   V : integer     { tegangan dalam volt }
   procedure PROSES1
   { Prosedur untuk "memproses" tahanan dan arus menjadi tegangan }

ALGORITMA
   input (R,A)
   PROSES1
   output (V)
```

```
procedure PROSES1
{ I.S : Diberikan harga R dan A yang telah terdefinisi}
{ F.S : Memproses R dan A sehingga dihasilkan V yaitu tegangan dengan rumus:
        V = R * A }

KAMUS LOKAL

ALGORITMA
   V ← R * A
```

Solusi 2 : Prosedur dengan parameter

```
Program VOLTAGE2
{ Program yang membaca tahanan dan arus, menghitung Voltage, dan mencetak
hasil perhitungan }

KAMUS
   R : integer     { tahanan dalam ohm }
   A : integer     { arus dalam ampere }
   V : integer     { tegangan dalam volt }

   procedure PROSES2 (input R, A : integer; output V : integer)
   { Prosedur untuk  "memproses" tahanan R dan arus A menjadi tegangan V}

ALGORITMA
   input (R,A)
   PROSES2 (R,A,V)
   output (V)
```

```
procedure PROSES2 (input R, A : integer; output V : integer)
{ I.S : Diberikan harga R dan A yang telah terdefinisi}
{ F.S : Memproses R dan A sehingga dihasilkan V yaitu tegangan dengan rumus:
        V = R * A }

KAMUS LOKAL

ALGORITMA
   V ← R * A
```

Catatan :

1. Prosedur dengan parameter lebih menjamin modularitas program. Sedapat mungkin semua prosedur diparametrisasi dengan baik.
2. Prosedur tanpa parameter bekerja dengan nama global. Hanya boleh dipakai untuk kasus yang sangat khusus, yaitu jika nama global merupakan *"universe"* (dunia, lingkungan) dari program, misalnya pada contoh mesin abstrak.
3. Prosedur tanpa parameter tidak boleh dipakai jika alasannya hanya karena pemrogram malas menuliskan parameter.

Contoh 2: PROSEDUR TUKAR

Prosedur untuk menukar dua harga yang disimpan dalam dua nama a dan b.

I.S. : Diberikan a = A dan b = B
F.S. : a = B dan b = A

```
Program TUKAR
{ Program yang membaca dua buah harga x dan y, menuliskan, menyimpannya,
kemudian menukarnya, dan menuliskan nilai setelah pertukaran }

KAMUS
    x,y : integer

    procedure PROCTUKAR (input/output a, b : integer)
    { Prosedur untuk menukar dua buah harga yang tersimpan dalam dua
       nama integer}
    { I.S : diberikan a=A dan b=B }
    { F.S : a=B dan b=A}

ALGORITMA
    input (x,y)
    PROCTUKAR ( x,y)
    output (x,y)
```

```
procedure PROCTUKAR (input/output a, b : integer)
{ I.S : diberikan a=A dan b=B }
{ F.S : a=B dan b=A }

KAMUS LOKAL
    Temp : integer

ALGORITMA
    Temp ← a    { Temp = a; a = a; b = b }
    a ← b       { Temp = a; a = b; b = b }
    b ← Temp    { Temp = a; a = b; b = a }
```

Contoh 3: PROSEDUR PUTAR3BIL

Contoh pemanfaatan prosedur yang pernah dipakai

Gunakan prosedur TUKAR untuk menulis prosedur yang "memutar" 3 buah nama integer.
Contoh : Jika a berisi 1, b berisi 2 dan c berisi 3, maka hasilnya :
a berisi 3, b berisi 1, dan c berisi 2.

```
procedure PUTAR3BIL (input/output a, b, c : integer)
{ I.S : a=A dan b=B, dan c=C }
{ F.S : a=C, b=A, c=B }
KAMUS LOKAL

ALGORITMA
   { I.S. a=A, b=B, c=C }
   PROCTUKAR (a,c)       { a = C; b= B, c=A }
   PROCTUKAR (b,c)       { a = C; b= A, c=B }
```

Contoh di atas adalah contoh sebuah prosedur yang memakai sebuah prosedur lain.

Catatan:

Contoh-contoh di atas tidak terlalu mewakili daya guna penulisan prosedur dengan nyata, pada bagian-bagian berikutnya akan terlihat lebih nyata manfaat penulisan prosedur, yang membuat:

- program mudah dibaca,
- prosedur yang sama dipakai berkali-kali sehingga mengurangi penulisan yang berulang,
- pengkodean yang independen.

Latihan Soal

1. Kerjakanlah kembali soal-soal pada bagian sebelumnya, dengan mendefinisikan prosedur, jika memang sesuai.
2. Apa beda prosedur dan fungsi?
3. Pada bahasa C, hanya ada "fungsi". Prosedur direalisasi dengan menuliskannya sebagai fungsi yang tidak menghasilkan harga. Apa pendapat Anda?

PENGULANGAN

Salah satu kemampuan komputer yang dapat dimanfaatkan adalah mengulang suatu instruksi, bahkan aksi, secara berulang-ulang dengan peformansi yang sama. Berbeda dengan manusia yang cenderung melakukan kesalahan jika melakukan hal yang sama (karena lelah atau bosan), komputer akan melakukan pengulangan dengan setia sesuai dengan perintah yang diberikan.

Pengulangan terdiri dari dua bagian:
- kondisi yang mengakibatkan pengulangan suatu saat berhenti, yang dinyatakan oleh sebuah ekspresi logik baik secara eksplisit maupun implisit,
- badan pengulangan, yaitu aksi yang harus diulang selama kondisi yang ditentukan untuk pengulangan masih dipenuhi.

Pengulangan harus berhenti, ini yang harus dijamin oleh pemrogram. Pada bab tentang "mengupas kentang" telah diberikan suatu contoh di mana pengulangan mungkin dilakukan terus menerus dan salah satu karena sifat algoritma yang harus dipenuhi adalah terjadi dalam selang waktu terbatas maka pengulangan yang terus menerus (*looping*) adalah algoritma yang salah.

Pengulangan yang terus-menerus harus dapat dideteksi pemrogram bahkan sebelum program dieksekusi oleh mesin, berdasarkan invariansi dari badan pengulangan tersebut.

Notasi pengulangan adalah salah satu notasi dasar dalam penulisan algoritma selain analisis kasus. Namun notasi tersebut hanya akan ada artinya jika dikenakan terhadap skema tertentu. Notasi pengulangan yang berdiri sendiri tidak ada artinya, bahkan menyesatkan karena pada hakekatnya notasi pengulangan hanyalah merupakan sebagian dari skema pengulangan yang akan dibahas pada bab-bab berikutnya.

Ada lima macam notasi pengulangan:

Pengulangan Berdasarkan Banyaknya Pengulangan

```
repeat n times

   <aksi>
   { n adalah nama informasi yang terdefinisi nilainya,
     bilangan bulat }
```

`<aksi>` akan diulang sebanyak n kali, dan bukan urusan pemrogram untuk mengelola pengulangan tersebut. Dengan hanya menyebutkan pengulangan tersebut, pengulangan pasti akan berhenti suatu saat.

Pengulangan Berdasarkan Kondisi Berhenti

```
repeat
   <aksi>
until <kondisi-berhenti>
```

<aksi> akan dihentikan jika kondisi-berhenti dipenuhi (bernilai *true*), akan diulang jika <kondisi-berhenti> belum tercapai. Badan pengulangan pada notasi ini (<aksi>) minimal akan dilakukan satu kali karena pada waktu eksekusi pengulangan yang pertama tidak ada dilakukan test terhadap <kondisi-berhenti>. Tes terhadap kondisi berhenti dilakukan setelah <aksi> dilaksanakan. Pengulangan ini berpotensi untuk menimbulkan "kebocoran" (ada <aksi> yang dileksekusi tanpa pernah diperiksa kondisi pelaksanaannya), jika ada kemungkinan bahwa seharusnya <aksi> tidak pernah boleh dilakukan untuk kasus yang tertentu.

Aksi sekuensial yang dilakukan adalah: <aksi>, tes <kondisi-berhenti>, [<aksi>, tes <kondisi-berhenti>, ..., dan seterusnya] diakhiri tes <kondisi-berhenti> yang bernilai *true*, yang menyebabkan pengulangan berhenti.

Pengulangan Berdasarkan Kondisi Ulang

```
while (<kondisi-pengulangan>) do
   <aksi>
{ Kondisi berhenti dicapai di titik program ini }
```

<aksi> akan dilakukan selama <kondisi-pengulangan> masih dipenuhi (bernilai *true*). Badan pengulangan (<aksi>) pada notasi ini mungkin tidak akan pernah dilakukan, karena sebelum aksi yang pertama dieksekusi dilakukan tes terhadap kondisi berhenti. Tes terhadap <kondisi-pengulangan> dilakukan setiap kali sebelum <aksi> dilaksanakan. Pengulangan ini berpotensi untuk menimbulkan aksi "kosong" (tidak pernah melakukan apa-apa karena pada tes yang pertama, <kondisi-pengulangan> tidak dipenuhi atau bernilai *false*).

Aksi sekuensial yang dilakukan adalah: tes <kondisi-pengulangan>, [<aksi>, tes <kondisi-pengulangan>, ..., dan seterusnya] diakhiri tes <kondisi-pengulangan> yang bernilai *false*, yang menyebabkan pengulangan berhenti.

Pengulangan Berdasarkan Dua Aksi

```
iterate

   <aksi-1>

stop (<kondisi-berhenti>)

   <aksi-2>

{ Kondisi berhenti dicapai di titik program ini }
```

Pengulangan ini seolah-olah adalah "gabungan" antara bentuk pengulangan kedua dan ketiga. Mekanisme yang dilakukan oleh pengulangan ini adalah dengan melakukan secara otomatis <aksi-1> pada eksekusi yang pertama kemudian dilakukan tes terhadap kondisi berhenti. Tergantung kepada kondisi berhenti yang dites:

- <aksi-2> akan diaktifkan dan kemudian <aksi-1> yang berikutnya diulang, atau
- pengulangan dihentikan karena efek neto dari <aksi-1> menghasilkan kondisi berhenti.

Pengulangan ini berguna untuk kasus-kasus di mana <aksi-2> merupakan hal yang harus dilakukan tergantung dari hasil <aksi-1>.

Aksi sekuensial yang dilakukan adalah: <aksi-1>, tes <kondisi-berhenti>, [<aksi-2>, tes <kondisi-berhenti>, ..., dan seterusnya] diakhiri tes <kondisi-berhenti> yang bernilai *true*, yang menyebabkan pengulangan berhenti.

Pengulangan Berdasarkan Pencacah

```
<nama-pencacah> traversal [<range-harga>]

   <aksi>

{ Catatan : <nama-pencacah> harus suatu type yang terdefinisi
  suksesor dan  predesesornya, setelah pelaksanaan pengulangan
  selesai, harga yang tersimpan pada <nama-pencacah> tidak
  terdefinisi: jika hendak dipakai, harus didefinisikan kembali }
```

<nama-pencacah> harus suatu type yang terdefinisi suksesor dan predesesornya, setelah pelaksanaan pengulangan selesai, harga yang tersimpan pada <nama-pencacah> tidak terdefinisi: jika hendak dipakai, harus didefinisikan kembali.

<aksi> akan dilakukan dengan memperhitungkan harga-harga dari <nama-pencacah> yang di-"jelajahi", dipakai satu per satu secara berturutan. Dengan memakai pengulangan ini, pemrogram tidak perlu melakukan operasi terhadap suksesor/predesesor karena setiap kali selesai melakukan <aksi>, otomatis mesin

akan melakukan operasi untuk mendapatkan suksesor dari harga yang sedang berlaku saat itu untuk nama.

Pengulangan otomatis berhenti setelah penjelajahan terhadap <nama-pencacah> sudah mencakup semua harga yang terdefinisi dalam range harga. Harga yang tersimpan pada <nama-pencacah> setelah pengulangan selesai dilaksanakan **tidak terdefinisi**, jika ingin dipakai harus didefinisikan kembali. Pengulangan ini biasanya dipakai jika harga yang tersimpan dalam <nama-pencacah> ingin dimanfaatkan dalam <aksi>, namum **tidak boleh diubah** karena akan mengacaukan urutan eksekusi yang dilakukan.

Pada bahasa pemrograman yang dapat dieksekusi mesin, nilai dan perubahan <nama-pencacah> dikontrol oleh pemroses bahasa. Harga yang tersimpan pada <nama-pencacah> setelah pengulangan selesai dilaksanakan tidak terdefinisi, jika ingin dipakai harus didefinisikan kembali.

Contoh

Berikut ini akan diberikan suatu contoh program kecil yang hasilnya sama, namun diimplementasi dengan ke bentuk pengulangan yang berbeda-beda.

Deskripsi persoalan:
Tuliskanlah sebuah program yang membaca sebuah nilai N (integer positif lebih besar daripada nol), dan menuliskan output nilai 1, 2, 3, 4, ..., s.d. N berderet ke bawah sebagai berikut:

```
1
2
3
....
...
N
```

Contoh:
Jika N = 3 maka outputnya adalah

```
1
2
3
```

Jika N = 1 maka outputnya adalah

```
1
```

```
Program TULISBIL1
{ Dibaca N > 0, menuliskan 1, 2, 3, ..., N berderet ke bawah, dengan bentuk
repeat N times }

KAMUS
   N, i : integer { bilangan yang akan ditulis }

ALGORITMA
   input (N)
   i ← 1
   repeat N times
      output (i)
      i ← i + 1
```

Catatan:

Sebenarnya untuk persoalan ini, bentuk pengulangan **repeat N times** kurang sesuai. Pemakaian yang sesuai misalnya jika harus menggambar segi empat, maka kita harus menggambarkan 4 sisi. Penggambaran setiap sisi inilah yang diulang 4 kali.

```
Program TULISBIL2
{ Dibaca N > 0, menuliskan 1, 2, 3, ..., N berderet ke bawah, dengan bentuk
repeat ... until ... }

KAMUS
    N, i : integer { bilangan yang akan ditulis }

ALGORITMA
    input (N)
    i ← 1
    repeat
        output (i)
        i ← i + 1
    until (i > N)
```

Penjelasan:

1. Dengan bentuk ini, **harus dijamin** bahwa nilai N yang dibaca sesuai dengan spesifikasi, yaitu **N > 0.** Jika nilai yang diberikan tidak sesuai dengan spesifikasi, maka akan tertulis angka 1. Untuk mengatasi hal ini dapat dilakukan misalnya bahwa nilai N yang dibaca "diulang" sehingga memenuhi persyaratan **N > 0**.
2. Nama yang didefinisikan sebagai i akan bernilai 1..N+1.
3. Anda dapat mendefinisikan i sebagai bilangan yang **sudah** ditulis dan memulai i dengan 0. Dalam hal ini nilai i adalah 0..N.

```
Program TULISBIL3
{ Dibaca N > 0, menuliskan 1, 2, 3, ..., N berderet ke bawah, dengan bentuk
while ... do ... }

KAMUS
    N, i : integer { bilangan yang akan ditulis }

ALGORITMA
    input (N)
    i ← 1
    while (i ≤ N) do
        output (i)
        i ← i + 1
    { i > N }
```

Penjelasan:

1. Dengan bentuk ini, nilai 1 belum tentu ditulis. Jika ternyata pengguna memberikan nilai N yang negatif atau 0, program tidak menuliskan apa-apa karena kondisi yang diperiksa pertama kali (i ≤ N) menghasilkan *false*.
2. Nama yang didefinisikan sebagai i akan bernilai 1..N+1.
3. Anda dapat mendefinisikan i sebagai bilangan yang **sudah** ditulis dan memulai i dengan 0. Dalam hal ini domain nilai i adalah 0..N.

```
Program TULISBIL4
{ Dibaca N > 0, menuliskan 1, 2, 3, ..., N berderet ke bawah, dengan bentuk
iterate }

KAMUS
   N, i : integer  { nilai yang akan ditulis }

ALGORITMA
   input (N)
   i ← 1
   iterate
      output (i)
   stop (i = N)
      i ← i + 1
   { i = N }
```

Penjelasan:

1. Dengan bentuk ini, nilai 1 pasti ditulis. Jika ternyata pengguna memberikan nilai N yang negatif atau 0, program akan menuliskan 1 kemudian berhenti.
2. Nama yang didefinisikan sebagai i akan bernilai 1..N, jika N positif.
3. Anda dapat mendefinisikan i sebagai bilangan yang **sudah** ditulis dan memulai i dengan 0. Dalam hal ini aksi di antara iterate ... stop adalah penambahan nilai i, sedangkan aksi sesudah kata *stop* adalah aksi penulisan.

```
Program TULISBIL5
{ Dibaca N > 0, menuliskan 1, 2, 3, ..., N berderet ke bawah, dengan bentuk
traversal }

KAMUS
   N, i : integer  { nilai yang akan ditulis }

ALGORITMA
   input (N)
   i traversal [1..N]
      output (i)
```

Penjelasan:

1. Bentuk ini ekivalen dengan bentuk iterate, untuk i [1..N], namun pertambahan nilai i ditangani secara implisit oleh pemroses bahasa.
2. Jika N yang diberikan oleh pengguna bernilai negatif, maka tidak terjadi aksi penulisan karena nilai i di luar domain [1..N].

Penutup

Ada bermacam-macam pengulangan, sebenarnya satu bentuk pengulangan dapat "diterjemahkan" menjadi bentuk yang lain dengan notasi algoritmik yang tersedia. Contohnya untuk persoalan menuliskan nilai 1...N di atas.

Instruksi pengulangan tidak dapat berdiri sendiri, melainkan harus disertai dengan instruksi-instruksi lain sebelum dan sesudah pengulangan.

Persoalannya adalah **memilih bentuk pengulangan yang benar dan tepat** untuk kelas persoalan tertentu. Pada kuliah algoritma dan pemrograman ini, pemilihan

bentuk pengulangan yang tepat merupakan salah satu objektif dari pelajaran dan merupakan inti dari "desain" algoritmik.

Tidak semua bahasa pemrograman yang ada menyediakan semua bentuk pengulangan di atas. Hal ini dapat dianalogikan dengan satu set pisau dapur yang terdiri dari puluhan macam pisau misalnya untuk memotong daging, sayur, buah. bahkan ada yang dirancang dengan sangat khusus untuk buah tertentu (*grape fruit*) karena mempermudah operasi pemotongan atau pengupasan yang dilakukan. Namun dalam keadaan darurat, ketika tidak semua pisau tersedia kita dapat memakai misalnya pisau sayur untuk memotong daging walaupun tidak semudah menggunakan pisau daging. Hal yang sama terjadi dengan pemilihan bentuk pengulangan jika bahasa pemrograman yang dipakai tidak menyediakan. Sebaiknya dalam hal ini yang dilakukan adalah memilih bentuk pengulangan yang tepat dan menterjemahkannya dalam instruksi yang tersedia pada bahasa eksekusi. Ini merupakan terjemahan desain algoritmik menjadi kode program.

SKEMA PEMROSESAN SEKUENSIAL

Definisi

Pemrosesan sekuensial adalah pemrosesan secara satu-persatu, dari sekumpulan informasi sejenis yang setiap elemennya dapat diakses dengan keterurutan tertentu (ada suksesor), jadi seakan-akan kumpulan elemen merupakan "deret" elemen. Elemen yang akan diproses dapat ber-type dasar (integer, real, character, boolean), tetapi dapat juga bertype komposisi (misalnya Point <x : real, y : real>).

Deret elemen dapat merupakan elemen yang dibaca satu per satu dari input *device*, nilai elemen suatu tabel atau matriks, disimpan dalam media penyimpanan sekunder (*file*), atau merupakan elemen list, dan sebagainya.

Kumpulan informasi itu disimpan sedemikian rupa, sehingga selalu dikenali melalui primitif yang mampu untuk memberikan:
- Elemen pertama (First_Elmt)
- Elemen yang siap diproses (Current_Elmt)
- Elemen yang diakses setelah Current_Elmt (Next_Elmt)
- Tanda akhir proses EOP

Skema sekuensial ini dibangun untuk mendapatkan suatu pola umum, sebab instruksi pengulangan seperti yang telah dibahas di bab sebelumnya hanya berfokus kepada **bentuk pengulangan,** belum mencakup semua abstraksi yang dibutuhkan untuk suatu proses sekuensial.

Spesifikasi Primitif

Primitif untuk skema sekuensial diterjemahkan menjadi prosedur dan nilai variabel sebagai berikut:

```
procedure First_Elmt
{ Aksi yang memberikan elemen pertama yang akan diproses secara
  sekuensial.
  I.S. : Sembarang
  F.S. : Current_Elmt berisi sebuah elemen yang akan diproses
}
```

```
procedure  Next_Elmt
{ Aksi yang memberikan elemen berikutnya dari Current_Elmt sesuai
  dengan aturan akses dari pengaturan deret elemen:
  - untuk elemen yang disimpan dalam memori internal secara kontigu
    (array), akses ke elemen berikutnya dapat dilakukan melalui
    indeks: indeks dari next element adalah suksesor dari Current-
    element.
  - untuk elemen yang disimpan dalam media sekunder - memori
    eksternal, akses ke elemen berikutnya dilakukan lewat suatu aksi
    yang tersedia (contoh : mesin karakter).
  - untuk elemen yang diatur sesuai dengan aturan lain, harus
    ditentukan fungsi yang memungkinkan teraksesnya elemen yang
    berikutnya.
  Realisasi dari aksi ini tergantung dari bagaimana informasi itu
  disimpan dalam memori:
  I.S. : Current_Elmt
  F.S. : Current_Elmt = Next_Elmt (Current-Elmt)
}
```

```
EOP : boolean
{ Bernilai true jika proses sekuensial harus dihentikan. Dapat
  diimplementasi sebagai fungsi element.

  Bagaimana EOP ini bernilai true?
  Ada dua macam model :
  1. Model dengan MARK : elemen terakhir adalah elemen "fiktif",
     sebetulnya bukan anggota elemen yang diproses, informasinya lain
     dari pada elemen yang diproses secara lumrah. Model dengan Mark
     justru lebih banyak dipakai dalam pemrograman praktis.
  2. Model tanpa MARK : elemen terakhir mengandung informasi yang
     memberitahukan bahwa elemen tsb. adalah elemen yang terakhir.
}
```

Contoh-contoh Elemen Proses Sekuensial

1. **Deret bilangan integer** [1, 2, 3, 4, 5, 6, 7, ..., N], N = 100
   Setiap elemen bertype integer
   First_Elmt akan mengakibatkan Current_Elmt = 1
   Next_Elmt akan mengakibatkan harga Current_Elmt yang baru = 2 jika Current_Elmt = 1
   EOP akan bernilai true untuk N = 100
   Perhatikanlah bahwa elemen-elemen berpola semacam itu **tidak perlu disimpan** semua sebab dapat dibangkitkan melalui "rumus": Next_Elmt dari Current_Elmt adalah Next_Elmt + 1; dengan syarat First_Elmt diketahui yaitu = 1.

2. **Pemrosesan data bertype bentukan:**
   type MHSNILAI : < NIM : integer > 0, MK : string, nilai : character ['A'..'E'] >
   Karena konteksnya mahasiswa ITB, perancang mengetahui bahwa domain dari mhs adalah sesuai dengan NIM mahasiswa ITB, dan tak satupun mahasiswa ITB bernomor pokok 9999999. Maka jika kita mempunyai sekumpulan informasi MHSNILAI yang dapat diakses secara sekuensial (tersimpan dalam tabel, atau dari sebuah *sequential file*) , dan misalnya data yang tersimpan adalah:
   <8689001,"IF221",'A'>, <8689002,"IF221",'A'>, <8689007,"IF221",'B'>,

<8690001,”IF223”,'E'>, <8690001,”IF222”,'C'>, <9999999,”XXXXX”,'A'>

Setiap elemen ber-type **MHSNILAI**

First_Elmt akan mengakibatkan Current_Elmt = <8689001, ”IF221”, 'A'>

Next_Elmt akan mengakibatkan harga Current_Elmt yang baru = <8689002, ”IF221”, 'A'>

EOP akan bernilai true untuk Current_Elmt = <9999999, ”XXXXX”, 'A'>

Perhatikanlah bahwa elemen-elemen berkarakteristik seperti ini **harus disimpan** atau dibaca satu per satu dari piranti masukan, sebab dari sebuah Current_Elmt, nilai Next_Elmt-nya tidak ada rumusnya.

Untuk semua teks algoritma yang merupakan skema pemrosesan sekuensial pada bab ini, dipakai prosedur-prosedur dan fungsi terdefinisi seperti yang dituliskan pada kamus sebagai berikut yang lebih ringkas daripada spesifikasi yang disertai catatan pada awal bab ini:

```
KAMUS
    procedure Inisialisasi   { Persiapan yang harus dilakukan
                               sebelum pemrosesan }
    procedure First_Elmt     { Memperoleh Current_Elmt yang pertama }
    procedure Proses_Current_Elmt    { Proses terhadap Current-Elmt }
    procedure Next_Elmt      { Memperoleh Current_Elmt yang baru }
    procedure Terminasi      { Proses yang harus dilakukan setelah
                               pemrosesan semua elemen selesai }
    EOP : boolean  { True jika keadaan sudah mencapai akhir proses:
                     - pada model dengan mark:
                       Current element = mark (elemen mark/fiktif)
                     - pada model tanpa maark :
                       Current element = elemen terakhir }
    procedure Proses_Kasus_Kosong    { Proses jika dijumpai kasus kosong:
                                       misalnya menuliskan pesan }
    procedure Proses_First_Elmt      { Proses khusus untuk elemen I }
```

Skema Pemrosesan Sekuensial dengan Mark

```
SKEMA PEMROSESAN DENGAN MARK
{ Tanpa penanganan kasus kosong secara khusus }
```

```
SKEMA
    Inisialisasi
    First_Elmt
    while not EOP do
        Proses_Current_Elmt
        Next_Elmt
    { EOP }
    Terminasi
```

```
SKEMA PEMROSESAN DENGAN MARK
{ Dengan penanganan kasus kosong }
```

```
SKEMA
    First_Elmt
    if (EOP) then
        Proses-Kasus-Kosong
    else
        Inisialisasi
        repeat
            Proses_Current-Elmt
            Next_Elmt
        until (EOP)
        Terminasi
```

```
SKEMA PEMROSESAN DENGAN MARK
{ Dengan kasus kosong, elemen pertama harus diproses khusus }
```

```
SKEMA
    First_Elmt
    if (EOP) then
        Kasus-Kosong
    else
        Inisialisasi
        Proses_First
        iterate
            Next_Elmt
        stop EOP
            Proses_Current-Elmt
        Terminasi
```

Skema Pemrosesan Sekuensial Tanpa Mark

```
SKEMA PEMROSESAN TANPA MARK
{ Karena tanpa mark, tak ada kasus kosong }
```

```
SKEMA
    Inisialisasi
    First_Elmt
    iterate
        Proses_Current-Elmt
    stop  EOP
        Next_Elmt
    Terminasi
```

```
SKEMA PEMROSESAN TANPA MARK
{ Karena tanpa mark, tak ada kasus kosong.
  Akses elemen pertama  tidak berbeda dengan akses Next_Elmt }
```

```
SKEMA
    Inisialisasi
    repeat
        Next_Elmt
        Proses_Current-Elmt
    until (EOP)
    Terminasi
```

```
SKEMA PEMROSESAN TANPA MARK
{ Karena tanpa mark, tak ada kasus kosong
  Pemrosesan khusus terhadap element pertama
  Pemrosesan elemen kedua dan seterusnya mungkin kosong }
```

```
SKEMA
    Inisialisasi
    First_Elmt
    Proses_First_Elmt
    while (not EOP) do
        Next_Elmt
        Proses_Current-Elmt
    { EOP }
    Terminasi
```

Studi Kasus Skema Pengulangan

Jumlah 1 s.d. N

Buatlah algoritma yang membaca sebuah bilangan bulat positif N, menuliskan: 1, 2, 3, ..., N dan menjumlahkan 1 + 2 + 3 + ... + N serta menuliskan hasil penjumlahan. Berikut ini adalah beberapa versi solusi dan penjelasannya:

Versi-1

```
Program SUMNBil1
{ Menjumlahkan 1+2+3+...N, dengan N yang dibaca }
{ Model tanpa mark, tanpa penanganan kasus kosong }

KAMUS
   i : integer       { bilangan yang akan diproses }
   N : integer > 0   { banyaknya bilangan yang dijumlahkan }
   Sum : integer     { jumlah }

ALGORITMA
   input (N); Sum ← 0    { Inisialisasi }
   i ← 1                 { First_Elmt: mulai dari 1 }
   iterate
      output (i)
      Sum ← Sum + i
   stop (i = N)          { EOP }
      i ← i + 1          { Next_Elmt }
   output (Sum)          { Terminasi }
```

Penjelasan:

1. Pengontrol pengulangan adalah bilangan integer i.
2. Analisis: pemrosesan sekuensial dari deret bilangan 1, 2, 3, ..., N.
   Model tanpa MARK,
   jika i adalah deret yang diproses, i adalah *current element*, i bernilai 1, 2, 3, ..., N
   EOP adalah jika i = N; First_Elmt : 1; Next_Elmt : i +1.
   Proses: menulis setiap bilangan, dan pada akhir proses menulis jumlah.
3. Program benar, dengan tambahan spesifikasi: N ≥ 1, sesuai dengan definisi domain i = 1..N.
4. Nilai i mentaati definisi domain 1..N

Versi-2

```
Program SUMNBil2
{ Menjumlahkan 1+2+3+...N, dengan N yang dibaca }
{ Model dengan mark, tanpa penanganan kasus kosong }

KAMUS
    i : integer      { bilangan yang akan dijumlahkan }
    N : integer >0   { banyaknya bilangan yang dijumlahkan }
    Sum : integer    { jumlah }

ALGORITMA
    input (N); Sum ← 0        { Inisialisasi }
    i ← 1                     { First_Elmt }
    while (i ≤ N) do
        output (i)
        Sum ← Sum + i
        i ← i + 1             { Next_Elmt }
    { i > N, i = N + 1, Sum = 1 + 2 + 3 + ... + N }
    output (Sum)              { Terminasi }
```

Penjelasan:

1. Pengontrol pengulangan adalah bilangan integer i.
2. Analisis: pemrosesan sekuensial dari deret bilangan 1, 2, 3, ..., N.
   Model dengan MARK,
   jika i adalah deret yang diproses, nilai i adalah Next element,
   i bernilai 1, 2, 3, ..., N
   EOP adalah jika i > N, yaitu i = N+1
   First_Elmt : 1;
   Next_Elmt : i +1.
   Proses : menulis setiap bilangan, dan pada akhir proses menulis jumlah.
3. Program benar, tanpa batasan nilai N. Jika N ≤ 0, terjadi kasus kosong.
4. Nilai i menjadi di luar domain, jika didefinisikan 1..N.

Versi-3

```
Program SUMNBil3
{ Menjumlahkan 1+2+3+...N, dengan N yang dibaca }
{ Model dengan mark, dengan  penanganan kasus kosong }

KAMUS
    i : integer      { bilangan yang akan dijumlahkan }
    N : integer >0   { banyaknya bilangan yang dijumlahkan }
    Sum : integer    { jumlah }

ALGORITMA
    input (N); SUM ← 0        { Inisialisasi }
    i ← 1                     { First_Elmt }
    repeat
        output (i)
        Sum ← Sum + i
        i ← i + 1             { Next_Elmt }
    until (i > N)
    { i > N ⇒ i = N+1, Sum = 1 + 2 + 3 + ... + N }
    output (Sum)              { Terminasi }
```

Penjelasan:

1. Pengontrol pengulangan adalah bilangan integer i.
2. Analisis: pemrosesan sekuensial dari deret bilangan 1, 2, 3, ..., N.
   Model dengan MARK (???)
   jika i adalah deret yang diproses, i adalah Next element, i bernilai 1,2,3..N
   EOP adalah jika Ii= N + 1; First_Elmt : 1; Next_Elmt : i +1.
   Proses : menulis setiap bilangan, dan pada akhir proses menulis jumlah.
3. Program benar, dengan tambahan spesifikasi: N ≥ 1, sesuai dengan definisi domain i = 1..N.
4. Nilai i di luar definisi domain 1..N.
5. Pemilihan skema tidak konsisten (bukan skema penanganan dengan kasus kosong!).

Versi-4

```
Program SUMNBil4
{ Menjumlahkan 1+2+3+...N, dengan N yang dibaca }
{ Model tanpa mark }

KAMUS
   i : integer        { bilangan yang akan dijumlahkan }
   N : integer >0     { banyaknya bilangan yang dijumlahkan }
   Sum : integer      { jumlah }

ALGORITMA
   input (N); SUM ← 0       { Inisialisasi }
   i traversal [1..N]
      output (i)
      Sum ← Sum + i
   { i = ?, Sum = 1 + 2 + 3 + ... + N }
   output (Sum)             { Terminasi }
```

Penjelasan:

1. Pengontrol pengulangan adalah bilangan integer i.
2. Analisis: pemrosesan sekuensial dari deret bilangan 1, 2, 3, ..., N.
   Model tanpa MARK.
   Jika i adalah nilai suku deret yang diproses, i adalah *current element*, i bernilai 1, 2, 3, ..., N.
   EOP adalah jika i = N; First_Elmt : 1; Next_Elmt : i +1.
   Proses: menulis setiap bilangan, dan pada akhir proses menulis jumlah.
3. Program benar, dengan tambahan spesifikasi : N ≥ 1, sesuai dengan definisi domain i = 1..N.
4. Nilai i sesuai definisi domain 1..N, namun setelah traversal tidak terdefinisi. Maka tidak boleh menggunakan nilai i setelah traversal berakhir.

Penjumlahan Deret Bilangan

Tuliskanlah sebuah program yang membaca nilai-nilai integer yang dibaca dari piranti masukan, dan menjumlahkan nilainya. Pemasukan nilai integer diakhiri dengan 9999.

```
Program JUMBilX
{ Menjumlahkan nilai-nilai X yang dibaca. Mark = 9999. }
{ Model dengan mark }

KAMUS
    X : integer     { sekumpulan bilangan integer yang dibaca untuk
                      dijumlahkan, pembacaan diakhiri dengan 9999 }
    Sum : integer   { jumlah }

ALGORITMA
    Sum ← 0         { Inisialisasi }
    input (X)       { First_Elmt }
    while (X ≠ 9999) do
        output (X)
        Sum ← Sum + X
        input (X)   { Next_Elmt }
    { Sum = X1 + X2 + ... + ... + Xi-1 }
    output (Sum)    { Terminasi }
```

Penjelasan:

1. Pengontrol pengulangan (*current element*) adalah nilai X.
2. Program ini **memakai skema yang benar**, tapi tidak menangani kasus kosong
3. Program tersebut benar untuk kasus kosong ataupun tidak kosong. Jika nilai yang diketikkan langsung 9999 (kasus kosong), mark tidak diproses.
4. Program tidak salah jika tidak pernah terjadi kasus kosong (nilai yang diketikkan minimal dilakukan satu kali).
5. Konklusi : program tersebut benar, dengan tambahan spesifikasi bahwa : jumlah bilangan untuk kasus kosong = 0.
6. Konklusi : **Skema yang dipilih cukup baik**.

Cacah Bilangan

Tuliskanlah sebuah program yang membaca nilai-nilai integer yang dibaca dari piranti masukan, dan **mencacah** banyaknya nilai integer yang diketikkan. Pemasukan nilai integer diakhiri dengan 9999.

Berikut ini adalah beberapa program solusinya, dengan penjelasannya:

```
Program CACAHBilX1
{ Mencacah (melakukan counting) nilai-nilai X yang dibaca. Mark = 9999. }
{ Model dengan mark }

KAMUS
   i : integer     { banyaknya bilangan integer yang sudah dibaca }
   X : integer     { nilai yang dibaca }

ALGORITMA
   i ← 0           { Inisialisasi }
   input (X)       { First_Elmt }
   while (X ≠ 9999) do
      output (X)
      i ← i + 1
      input (X)    { Next_Elmt }
   { X = 9999, i adalah banyaknya bilangan yang sudah dibaca }
   output (i)      { Terminasi }
```

Penjelasan :

1. Pengontrol pengulangan (*current element*) adalah nilai X yang dibaca.
2. Program ini **memakai skema yang benar**, tapi tidak menangani kasus kosong.
3. Program tersebut benar untuk kasus kosong ataupun tidak kosong. Jika nilai yang diketikkan langsung 9999 (kasus kosong), mark tidak diproses.
4. Program tidak salah jika tidak pernah terjadi kasus kosong (nilai yang diketikkan minimal dilakukan satu kali).
5. Hasil pencetakan i benar, inisialisasi sesuai dengan definisi.
6. Konklusi: program tersebut benar, dengan tambahan spesifikasi bahwa : banyaknya bilangan yang diketikkan adalah 0 untuk kasus kosong
7. Konklusi: **Skema yang dipilih baik, definisi i tepat**.

```
Program CACAHBilX2
{ Mencacah (melakukan counting) nilai-nilai X yang dibaca. Mark = 9999 }
{ Model dengan mark }

KAMUS
   i : integer     { banyaknya bilangan integer yang AKAN dibaca }
   X : integer     { nilai yang dibaca }

ALGORITMA
   i ← 1           { Inisialisasi }
   input (X)       { First_Elmt }
   while (X ≠ 9999) do
      output (X)
      i ← i + 1
      input (X)    { Next_Elmt }
   { X = 9999, i adalah banyaknya bilangan yang AKAN dibaca }
   output (i-1)    { Terminasi }
```

Penjelasan :

1. Pengontrol pengulangan (*current element*) adalah nilai X.
2. Program ini **memakai skema yang benar**, tapi tidak menangani kasus kosong
3. Program tersebut benar untuk kasus kosong ataupun tidak kosong. Jika nilai yang diketikkan langsung 9999 (kasus kosong), mark tidak diproses.
4. Program tidak salah jika tidak pernah terjadi kasus kosong (nilai yang diketikkan minimal dilakukan satu kali).
5. Hasil pencetakan i benar, inisialisasi sesuai dengan definisi. Namun tidak natural karena pada inisialisasi, bilangan yang "akan" dibaca adalah 1.
6. Konklusi: program tersebut benar, dengan tambahan spesifikasi bahwa: banyaknya bilangan yang diketikkan adalah 0 untuk kasus kosong.
7. Konklusi: **Skema yang dipilih baik**, tapi **definisi i kurang tepat**.

```
Program CACAHBilX3
{ Mencacah (melakukan counting) nilai-nilai X yang dibaca. Mark = 9999 }
{ Model dengan mark }

KAMUS
   i : integer     { banyaknya bilangan integer yang AKAN dibaca }
   X : integer     { nilai yang dibaca }

ALGORITMA
   i ← 1           { Inisialisasi }
   input (X)       { First_Elmt }
   while (X ≠ 9999) do
      output (X)
      input (X)    { Next_Elmt }
      i ← i + 1
   { X = 9999, i adalah banyaknya bilangan yang AKAN dibaca }
   output (i-1)    { Terminasi }
```

Penjelasan :

1. Pengontrol pengulangan (*current element*) adalah nilai X.
2. Program ini **memakai skema yang benar**, tapi tidak menangani kasus kosong.
3. Program tersebut benar untuk kasus kosong ataupun tidak kosong. Jika nilai yang diketikkan langsung 9999 (kasus kosong), mark tidak diproses.
4. Program tidak salah jika tidak pernah terjadi kasus kosong (nilai yang diketikkan minimal dilakukan satu kali).
5. Hasil pencetakan i benar, inisialisasi sesuai dengan definisi. Namun tidak natural karena pada inisialisasi, bilangan yang "akan" dibaca adalah 1.
6. Konklusi: program tersebut benar, dengan tambahan spesifikasi bahwa : banyaknya bilangan yang diketikkan adalah 0 untuk kasus kosong.
7. Konklusi: **Skema yang dipilih baik**, tapi **definisi i kurang tepat dan penambahan nilainya** tidak pada skema yang benar (sesudah *next element*).

Latihan soal:

Coba implementasikan skema di atas dengan *repeat* dan *iterate*, berikan komentar.

Jumlahkan dan Cacah Bilangan

Tuliskanlah sebuah program yang membaca sekumpulan nilai integer yang diketikkan lewat piranti masukan, dan sekaligus melakukan:
- penjumlahan dari nilai integer yang dibaca,
- mencacah banyaknya nilai integer yang dibaca.

Pada akhir program, harus dituliskan jumlah dan banyaknya integer yang dibaca.

Berikut ini adalah lima buah versi program, dengan penjelasannya:

```
Program SUMNBilX1
{ Menjumlahkan dan mencacah (melakukan counting) nilai-nilai X yang
dibaca. Mark = 9999 }

KAMUS
   i : integer      { banyaknya nilai integer yang akan dibaca }
   X : integer      { sekumpulan bilangan integer yang dibaca, diakhiri
                      dengan 9999}
   N : integer > 0 { banyaknya bilangan yang dijumlahkan }
   Sum : integer    { jumlah }

ALGORITMA
   i ← 1; SUM ← 0              { Inisialisasi }
   input (X)                   { First_Elmt }
   while (X ≠ 9999) do
      output (X)
      Sum ← Sum + X
      i ← i + 1
      input (X)                { Next_Elmt }
   { i = bilangan ke... yang akan dibaca,
     Sum = X1 + X2 + ... + ... + Xi-1 }
   output ("Jumlah : ", Sum)    { Terminasi }
   output ("Banyaknya bilangan  : ", i - 1)
```

Penjelasan :

1. Pengontrol pengulangan (Current element) adalah nilai X.
2. Program ini **memakai skema yang benar**, tapi tidak menangani kasus kosong
3. Program tersebut benar untuk kasus kosong ataupun tidak kosong. Jika nilai yang diketikkan langsung 9999 (kasus kosong), mark tidak diproses.
4. Program tidak salah jika tidak pernah terjadi kasus kosong (nilai yang diketikkan minimal dilakukan satu kali).
5. Hasil pencetakan i benar, inisialisasi sesuai dengan definisi. Namun tidak natural karena pada inisialisasi, bilangan yang "akan" dibaca adalah 1.
6. Konklusi: program tersebut benar, dengan tambahan spesifikasi bahwa :
   - banyaknya bilangan yang diketikkan adalah 0 untuk kasus kosong.
   - jumlah bilangan untuk kasus kosong = 0.
7. Konklusi: **Skema yang dipilih baik**, tapi **definisi i kurang tepat**.

```
Program SUMNBilX2
{ Menjumlahkan dan mencacah (melakukan counting)  nilai-nilai X yang dibaca.
Mark = 9999 }

KAMUS
   i : integer     { banyaknya nilai integer sudah dibaca }
   X : integer     { sekumpulan bilangan integer yang dibaca, diakhiri
                     dengan 9999 }
   N : integer > 0 { banyaknya bilangan yang dijumlahkan }
   Sum : integer   { jumlah }

ALGORITMA
   i  ← 0;  SUM ← 0           { Inisialisasi }
   input (X)                  { First_Elmt }
   while (X ≠ 9999) do
      output (X)
      Sum ← Sum + X
      i ← i + 1
      input (X)               { Next_Elmt }
   { i = bilangan ke... yang sudah dibaca,
     Sum = X1 + X2 + ... + ... + Xi-1 }
   output ("Jumlah : ", Sum)  { Terminasi }
   output ("Banyaknya bilangan  : ", i)
```

Penjelasan:

1. Pengontrol pengulangan (*current element*) adalah nilai X.
2. Program ini **memakai skema yang benar**, tapi tidak menangani kasus kosong.
3. Program tersebutbenar baik untuk kasus kosong ataupun tidak kosong. Jika nilai yang diketikkan langsung 9999 (kasus kosong), mark tidak diproses.
4. Program tidak salah jika tidak pernah terjadi kasus kosong (nilai yang diketikkan minimal dilakukan satu kali).
5. Konklusi : program tersebut benar, dengan tambahan spesifikasi bahwa:
   - banyaknya bilangan yang diketikkan adalah 0 untuk kasus kosong.
   - jumlah bilangan untuk kasus kosong = 0.

```
Program SUMNBilX3
{ Menjumlahkan dan mencacah (melakukan counting) nilai-nilai X yang dibaca.
Mark = 9999 }

KAMUS
   i : integer     { banyaknya nilai integer sudah dibaca }
   X : integer     { sekumpulan bilangan integer yang dibaca, diakhiri
                     dengan 9999 }
   Sum : integer   { jumlah }

ALGORITMA
   i ← 0; SUM ← 0              { Inisialisasi }
   input (X)                   { First_Elmt }
   repeat
      output (X)
      Sum ← Sum + X
      i ← i + 1
      input (X)                { Next_Elmt }
   until (X = 9999)
   { i = banyaknya bilangan yang sudah dibaca,
     Sum = X1 + X2 + ... + Xi }
   output ("Jumlah : ", Sum)   { Terminasi }
   output ("Banyaknya bilangan  : ", i)
```

Penjelasan :

1. Pengontrol pengulangan (Current element) adalah nilai X.
2. Program ini **memakai skema yang salah**, bukan salah satu dari skema yang diberikan. Bentuk pengulangan *repeat* mengharuskan badan pengulangan dilakukan minimal satu kali.
3. Program tersebut salah untuk kasus kosong. Jika nilai yang diketikkan langsung 9999 (kasus kosong), mark akan diproses sehingga program menjadi **salah**.
4. Program tidak salah jika tidak pernah terjadi kasus kosong (nilai yang diketikkan minimal dilakukan satu kali).
5. Konklusi: **program tersebut salah**.

```
Program SUMNBilX4
{ Menjumlahkan dan mencacah (melakukan counting) nilai-nilai X yang dibaca.
Mark = 9999. Dengan penanganan kasus kosong }

KAMUS
   i : integer     { banyaknya nilai integer sudah dibaca }
   X : integer     { sekumpulan bilangan integer yang dibaca, diakhiri
                     dengan 9999 }
   N : integer >0  { banyaknya bilangan yang dijumlahkan }
   Sum : integer   { jumlah }

ALGORITMA
   input (X)                      { First_Elmt }
   if (X ≠ 9999) then
      output ("Kasus kosong: yg diketik langsung 9999")
   else
      i ← 0; SUM ← 0              { Inisialisasi }
      repeat
         output (X)
         Sum ← Sum + X
         i ← i + 1
         input (X)                { Next_Elmt }
      until (X = 9999)
      { i = banyaknya bilangan yang sudah dibaca,
        Sum = X1 + X2 + ... + Xi }
      output ("Jumlah : ", Sum)  { Terminasi }
      output ("Banyaknya bilangan  : ", i)
```

Penjelasan:

1. Pengontrol pengulangan (*current element*) adalah nilai X.
2. Program ini **memakai skema yang tepat**, dengan penanganan kasus kosong.
3. Jika nilai yang diketikkan langsung 9999, kasus kosong, jika bukan berarti minimal ada satu nilai yang dibaca.
4. Semua invarian dipenuhi. Definisi dan inisialisasi i tepat artinya.

```
Program SUMNBilX5
{ Menjumlahkan dan mencacah (melakukan counting)  nilai-nilai X yang dibaca.
Mark = 9999. Perhatikan bahwa model ini tanpa Mark. }

KAMUS
    i : integer     { banyaknya nilai integer sudah dibaca }
    X : integer     { sekumpulan bilangan integer yang dibaca, diakhiri
                      dengan 9999 }
    Sum : integer   { jumlah }

ALGORITMA
    i ← 1; SUM ← 0       { Inisialisasi }
    iterate
        input (X)        { First_Elmt, juga Next_Elmt }
        output (X)
        Sum ← Sum + X
    stop (X = 9999)
        i ← i + 1
    until (X = 9999)
    { i = banyaknya bilangan k yang sudah dibaca,
      Sum = X1 + X2 + ... + Xi }
    output (Sum)         { Terminasi }
```

Penjelasan:

1. Pengontrol pengulangan (*current element*) adalah nilai X.
2. Program ini **salah**, jika nilai yang diketikkan langsung 9999 (kasus kosong), karena Mark diproses (dijumlahkan).
3. Skema yang dipakai adalah skema tanpa mark, jadi bukan skema yang tepat untuk persoalan ini.

HUBUNGAN BERULANG

(Reccurence Relation)

Bagian ini menyajikan contoh-contoh dan pola pemakaian skema pengulangan untuk persoalan deret yang rumusnya dapat dinyatakan dalam suatu hubungan berulang *(reccurence relation).* Bagian ini sekaligus menunjukkan keterbatasan analisis dan penulisan algoritma, yaitu yang menyangkut masalah ketelitian representasi bilangan riil pada komputer. Untuk melihat efek tersebut, harus dilakukan eksekusi pada mesin.

Contoh 1:

Hitunglah S = 1 - 1/2 + 1/3 - 1/4 + ... + 1/999 - 1/1000
Ide pertama yang muncul adalah menyatakan suku ke-i sebagai:

$$S_i = (-1)^{i+1}/i$$

Berikut ini adalah beberapa versi solusi, dengan penjelasannya:

```
Program HITUNGDERET1
{ Menghitung deret S = 1 - 1/2 + 1/3 - 1/4 + ... + 1/999 - 1/1000 }

KAMUS
    constant N : integer = 1000
    i : integer      { indeks suku berikut yang akan dihitung }
    S : real ≥ 0.0   { Jumlah deret }

ALGORITMA
    S ← 0     { i = 0, S = 0 }
    i  ← 1    { suku berikutnya yang akan dihitung }
    while (i ≤ N) do
        { S = 1 - 1/2 + 1/3 - 1/4 + ... + (-1)^i/(i-1) }
        S ← S + (-1)^(i+1)/i
        i  ← i + 1
    { i = N + 1 and S = 1 - 1/2 + 1/3 - 1/4 + ... + 1/999 - 1/N, N = 1000 }
    output (S)
```

Penjelasan :

1. Pengontrol pengulangan adalah sebuah bilangan integer i, i adalah *next element*.
2. Perhatikan inisialisasi. Invarian dari i dan S.
3. Program menghasilkan nilai jumlah deret nol, jika konstanta N bernilai negatif, N ≤ 0, dengan tambahan spesifikasi: Jumlah deret adalah 0 untuk N ≤ 0.
4. Jika diimplementasikan dalam salah satu bahasa pemrograman, harus diperhatikan konversi type sebab misalnya 1/2 adalah ekspresi integer, yang hasilnya adalah integer.

```
Program HITUNGDERET2
{ Menghitung deret S = 1 - 1/2 + 1/3 - 1/4 + ... + 1/999 - 1/1000 }

KAMUS
    constant N : integer = 1000
    i : integer      { indeks suku berikut yang akan dihitung }
    S : real ≥ 0.0   { Jumlah deret }

ALGORITMA
    S ← 1     { i = 1, S = 1 }
    i ← 2     { suku berikutnya yang akan dihitung }
    while (i ≤ N) do
        { S = 1 - 1/2 + 1/3 - 1/4 + ... + (-1)^i/(i-1)}
        S ← S + (-1)^(i+1)/i
        i ← i + 1
    { i = N + 1 and S = 1 - 1/2 + 1/3 - 1/4 + ... + 1/999 - 1/N, N = 1000 }
    output (S)
```

Penjelasan:

1. Pengontrol pengulangan adalah sebuah bilangan integer i, i adalah *next element*.
2. Perhatikan inisialisasi. Invarian dari i dan S. Program benar dengan tambahan spesifikasi : N $\geq$ 1.

```
Program HITUNGDERET3
{ Menghitung deret S = 1 - 1/2 + 1/3 - 1/4 + ... + 1/999 - 1/1000 }

KAMUS
    constant N : integer = 1000
    i : integer      { indeks suku berikut yang akan dihitung }
    TANDA : [-1,1]   { tanda suku deret}
    S : real ≥ 0.0   { Jumlah deret}

ALGORITMA
    TANDA ← 1
    S ← 1     { i = 1, TANDA = 1, S = 1 }
    i ← 2     { suku berikutnya yang akan dihitung }
    while (i ≤ N) do
        { S = 1 - 1/2 + 1/3 - 1/4 + ... + tanda/(i-1) }
        TANDA ← -TANDA
        S ← S + TANDA/i
        i ← i + 1
    { i = N + 1 and S = 1 - 1/2 + 1/3 - 1/4 + ... + 1/999 - 1/N, N = 1000 }
    output (S)
```

Penjelasan:

1. Merupakan modifikasi dari versi 2. Tampaknya, algoritma versi 1 sederhana, tetapi sebenarnya tidak efisien karena mengandung proses pemangkatan yaitu perhitungan (-1)^(i + 1). Karena proses pemangkatan ini gunanya hanya untuk mengubah tanda dari satu suku ke suku berikutnya, maka harus dipikirkan cara lain yang lebih bagus untuk mengatur perubahan tanda ini. Pada algoritma versi-2 ini, dipakai sebuah variabel tambahan **TANDA** yang harganya berubah-ubah, yaitu +1 atau -1.
2. Sebuah *loop* "while" di atas, merupakan penggabungan dari dua loop yang dijalankan secara bersamaan, yaitu *loop* dengan elemen pengontrol i dan tanda.

```
Program HITUNGDERET4
{ Menghitung deret S = 1 - 1/2 + 1/3 - 1/4 + ... + 1/999 - 1/1000 }

KAMUS
    constant N : integer = 1000
    i : integer      { indeks suku terakhir yang sudah ditambahkan ke S}
    TANDA : [-1,1]  { tanda suku deret }
    S : real ≥ 0.0  { Jumlah deret }

ALGORITMA
    TANDA ← -1
    S ← 1/2   { i = 2, TANDA = -1, S = 1 - 1/2 }
    i ← 2     { suku terakhir yang sudah ditambahkan ke S }
    while (i < N) do
        { Kondisi untuk mengakhiri iterasi, suku 1/1000 sudah terhitung }
        { S = 1 - 1/2 + 1/3 - 1/4 + ... + tanda/i }
        i  ← i + 1
        TANDA ← -TANDA
        S ← S + TANDA/i
    { i = N and S = 1 - 1/2 + 1/3 - 1/4 + ... + 1/999 - 1/N, N = 1000 }
    output (S)
```

Penjelasan:

1. Pengontrol pengulangan adalah sebuah bilangan integer i, i adalah *current element*.
2. Perhatikan inisialisasi: Invarian dari i dan S.
3. Program benar dengan tambahan spesifikasi $N \geq 2$.

Latihan:

1. Carilah algoritma-algoritma lain untuk soal ini (paling tidak, ada empat solusi lain).
2. Buatlah program komputer, untuk masalah ini berdasarkan algoritma-algoritma yang berbeda. Bandingkan hasilnya. Pakailah ketelitian yang berbeda-beda pula untuk setiap algoritma. Apa yang terjadi jika algoritma di atas diterjemahkan secara naif dan diprogram dalam bahasa pemrograman yang dapat dieksekusi?

Contoh 2: FAKTORIAL

Hitunglah n!
Dari definisi:

$0! = 1$
$1! = 1$
$2! = 2 \times 1! = 2 \times 1 = 2$
$3! = 3 \times 2! = 3 \times 2 = 6$
$4! = 4 \times 3! = 4 \times 6 = 24$
...
$n! = n \times (n-1)! = n \times (n-1) \times ... \times 3 \times 2 \times 1$

Hubungan berulang:

$$F_o = 1$$
$$F_i = i * F_{i-1}$$

Solusi 1:

```
Program FAKTORIAL1
{ Diberikan n, dihitung n! }

KAMUS
   i : integer ≥ 1    { indeks F berikut yang akan dihitung }
   n : integer ≥ 0    { bilangan yang dihitung faktorialnya }
   F : integer ≥ 1    { harga faktorial yang dihitung }

ALGORITMA
   input (n)
   F ← 1     { F_o = 1 }
   i ← 1     { indeks F berikutnya yang akan dihitung }
   while (i ≤ n) do
      { F = (i-1)! }
      F ← i * F
      i ← i + 1
   { i = n + 1 and F = n! }
   output (F)
```

Solusi 2:

```
Program FAKTORIAL2
{ Diberikan n, dihitung n!, kondisi berhenti berdasarkan loop invariant }

KAMUS
   i : integer ≥0  { indeks F terakhir dihitung }
   n : integer ≥0  { bilangan yang dihitung faktorialnya }
   F : integer ≥ 1 { harga faktorial yang dihitung }

ALGORITMA
   input(n)
   F ← 1     { F_o }
   i ← 0     { indeks F, untuk nilai F yang telah dihitung }
   while (i < n)  do
      { F = i! }
      i ← i + 1
      F ← i * F
   { i = n and F = n! }
   output (F)
```

Contoh 3. HITUNG EKSPONEN

Hitung exp(x) dengan pendekatan:

$$\lim_{n \to \infty} S_n;\ S_n = 1 + x + x^2/2! + x^3/3! + ... + x^n/n!\ ,\ n \geq 0$$

Deret di atas adalah deret aditif dengan suku umum : $T_i = x^{i-1}/i!\quad i \geq 0$

Dengan menerapkan skema umum yang telah dikenal, didapat algoritma sebagai berikut:

```
Program HITUNGEXP1
{ Menghitung exp (x) ≈ 1 + x + x^2/2! + x^3/3! + ..... x^n/n! }
```

```
KAMUS
   T : real          { Suku terakhir dihitung}
   S : real          { Jumlah deret }
   i : integer ≥ 0 { indeks suku yang terakhir dihitung }
   { P adalah sebuah Predikat, dengan parameter S, T, dan i }
```

```
ALGORITMA
   T ← 1 { T0 }
   S ← T
   i ← 0
   while not P(S,T,i) do
      { Kondisi berhenti P(S,T,i) akan diperinci lebih lanjut }
      i ← i + 1
      T ← x^i / FACT(i)    { FACT(n) adalah fungsi untuk menghitung n! }
      S ← S + T
   output (S)
```

Contoh 4. HITUNG SINUS

Hitung sin(x) dengan pendekatan:

$$\lim_{n \to \infty} S_n;\ S_n = x - x^3/3! + x^5/5! - ... + (-1)^{n-1}x^{2n-1}/(2n-1)!,\ n \geq 1$$

$\sin(x) \approx x - x^3/3! + x^5/5! - ...$

Suku umum dapat ditulis sebagai: $t_i = (-1)^{i-1}x^{2i-1}/(2i-1)$

```
Program HITUNGSIN1
{ Menghitung sin (x) ≈ x - x^3/3! + x^5/5! - ... }

KAMUS
   T : real          { Suku terakhir dihitung }
   S : real          { Jumlah deret }
   i : integer ≠ 0 { indeks suku yang terakhir dihitung }

ALGORITMA
   i ← x { T1 }
   S ← T
   i ← 1
   while not P(S,T,i) do
      { Kondisi berhenti P(S,T,i) akan diperinci lebih lanjut }
      i ← i + 1
      T ← ((-1)^(i-1)) * (x^(2*(i-1))) / FACT(2*(i-1))
      { FACT (n) adalah fungsi untuk menghitung n! }
      S ← S + T   { Deret aditif }

   output (S)
```

Seperti pada contoh S = 1 - 1/2 + 1/3 - ..., algoritma-algoritma ini menyangkut perhitungan suku ke-i secara langsung yang sangat tidak efisien dan memakan waktu. Pada hampir semua kasus, kita dapat memanfaatkan hasil perhitungan sebelumnya. Maka mula-mula kita definisikan hubungan berulang untuk setiap suku:

T_o = harga awal

$T_i = f(T_{i-1}),\quad i > 0$

atau

T_o = harga awal

$T_i = f(T_{i-1}, i),\ i > 0$

Sehingga skema umum menjadi:

Skema umum 1:

```
Program HITUNGDERET1
{ Menghitung deret S = t0+t1+t2+...+tn atau S = t0*t1*t2*...*tn }

KAMUS
   T : real  { Suku/faktor terakhir dihitung }
   S : real  { Jumlah deret aditif/multiplikatif }

ALGORITMA
   T ← t0
   S ← T
   while not P(S,T) do
      T ← f(T)
      { f(T) adalah fungsi yang menghitung harga T baru berdasarkan T yang
        lama }
      S ← g(S,T)
      { g(S,T) adalah fungsi yang mengubah harga S berdasarkan S yang lama
        dan T }

   { S adalah hasil yang diinginkan}
```

Skema umum 2:

```
Program HITUNGDERET2
{Menghitung deret S = t 0 + t 1 + t 2 +... +tn  atau  S = t 0 * t 1* t 2 *
... * tn }

KAMUS
   T : real         { Suku/faktor terakhir dihitung }
   S : real         { Jumlah deret aditif/multiplikatif }
   i : integer ≠ 0 { indeks suku/faktor yang terakhir dihitung }

ALGORITMA
   T ← t0
   S ← T
   i ← 0
   while not P(S,T) do
      { loop invariant: S = t0 + t1 + t2 + ... + ti
                        atau S = t0 * t1 * t2 * ... * ti }
      { i adalah indeks suku terakhir yang sudah termasuk dalam S }
      i ← i + 1
      T ← f(T,i)
      { f(T) adalah fungsi yang menghitung harga T yang baru
        berdasarkan T yang lama dan i }
      S ← g(S,T,i)
      { g(S,T,i) adalah fungsi yang mengubah harga S
        berdasarkan S yang lama, T dan i}

   { S adalah hasil yang diinginkan}
```

Maka untuk algoritma hubungan berulang menghitung pendekatan exp(x):

$$\exp(x) \approx \lim_{n \to \infty} 1 + x + x^2/2! + x^3/3! + \ldots + x^n/n!$$

Kita dapatkan:

Hubungan berulang:

$T_o = 1$

$T_i = T_{i-1} * x/i$

Kondisi berhenti:

$|T| \leq \varepsilon$

Sehingga algoritmanya menjadi:

```
Program HITUNGEXP2
{ Menghitung exp (x) ≈ 1 + x + x^2/2! + x^3/3! + ... + x^n/n!  }

KAMUS
   T : real          { Suku terakhir dihitung }
   S : real          { Jumlah deret }
   i : integer ≠ 0 { indeks suku yang terakhir dihitung }

ALGORITMA
   T ← 1 { T0 }
   S ← T
   i ← 0
   while (abs(T) > ε) do
      i ← i + 1
      T ← T * x/i
      S ← S + T
   output (S)
```

Dan untuk perhitungan sin(x) dengan:

$$T_i = (-1)^{2i-1} * x^{2i-1}/(2i-1)!$$

lebih disukai suatu variabel k sehingga

$$T_i = -T_{i-1} * x^2/k_i(k_i-1)$$
$$k_i = k_{i-1} + 2$$
$$T_o = x$$
$$k_o = 1$$

Kondisi berhenti: $|T| \leq \varepsilon * |S|$

Algoritmanya menjadi:

```
Program HITUNGSIN2
{ Menghitung sin (x) ≈ x - x^3/3! + x^5/5! - ... }

KAMUS
   T : real           { Suku terakhir dihitung }
   S : real           { Jumlah deret }
   k : [1,3,5,...]    { indeks suku yang terakhir dihitung }

ALGORITMA
   k ← 1
   T ← x { T1 }
   S ← T
   while (abs(T) > ε * abs(S)) do
      k ← k + 2
      T ← -T * x * x/ (k*(k-1))
      S ← S + T   { deret aditif }
   { abs(T) ≤ε * abs(S) → suku berikutnya sudah dapat diabaikan: }

   output (S)
```

Latihan Soal

1. Hitung cos(x) dengan pendekatan:
   $S = 1 - x^2/2! + x^4/4! - ... + (-1)^n x^{2n} /(2n)!, n \geq 0$
   dan kondisi berhenti: $|T| \leq \varepsilon * |S|$

2. Buat suatu algoritma untuk menghitung cos(x) maupun sin(x) tergantung pada sebuah parameter p, p = 1 untuk cos(x) atau p = 2 untuk sin(x) dengan memperhatikan bahwa kedua rumus untuk menghitung sin(x) dan cos(x) sangat mirip.

3. Hitung pendekatan $\Leftrightarrow {}_0{}^n \exp(-x^2)\, dx$ dengan pendekatan:
   $x - x^3/3*1! + x^5/5*2! - x^7/7*3! + ...$

4. Hitunglah konversi string "12345" menjadi nilai integer basis 10, atau sebuah string dengan basis bukan sepuluh, misalnya biner yaitu "000011111001" menjadi nilai integer basis 10 tanpa mengetahui posisinya, yaitu dengan memanfaatkan nilai sebelumnya yang sudah dihitung.

ARRAY, TABEL KONTIGU

Type array adalah type yang mengacu kepada sebuah atau sekumpulan elemen melalui indeks. Elemen dari array dapat diakses langsung jika dan hanya jika indeks terdefinisi (ditentukan harganya dan sesuai dengan domain yang didefinisikan untuk indeks tersebut). Array biasanya disebut juga sebagai t**abel, vektor,** atau **larik**. Nama suatu array diasosiasikan dengan banyak nilai elemennya yang disimpan dalam nama tersebut.

Struktur data ini dipakai untuk merepresentasikan sekumpulan informasi yang bertype sama, dan disimpan dengan urutan yang sesuai dengan definisi indeks **secara kontigu dalam memori** komputer. Karena itu indeks harus suatu type yang mempunyai keterurutan (ada suksesor dan predesesor), misalnya type integer, karakter.

Jika indeksnya adalah integer, maka keterurutan indeks sesuai dengan urutan integer (suksesor adalah plus satu, predesesor adalah minus satu). Jika indeksnya ditentukan sesuai dengan enumerasi (misalnya bertype pada karakter), maka keterurutan indeks ditentukan sesuai dengan urutan enumerasi.

Contoh:

```
KAMUS
    TabNamaHari      : array [1..7] of string
    TabJumlahHari    : array [1..12] of integer

    type Point       : < x : integer, y : integer >
    type Indeks      : integer [1..10]
    TabTitikSurvey   : array [Indeks] of  Point

    TabFREK          : array ['A'..'Z'] of integer
```

Domain:
- Domain array sesuai dengan pendefinisian indeks
- Domain isi array sesuai dengan jenis array

Konstanta:
- Konstanta untuk seluruh array tidak terdefinisi,
- Konstanta hanya terdefinisi jika indeks dari array terdefinisi

Cara mengacu sebuah elemen: melalui indeks

TabNamaHari$_i$, jika i terdefinisi

TabNamaHari$_7$

TabJumlahHari$_3$

Contoh Pemakaian Array

Kasus 1: NAMA HARI

Nama hari dalam minggu akan direpresentasi sebagai array sebagai berikut, dan harus dituliskan sebuah algoritma yang membaca hari ke berapa [1..7], kemudian menuliskan nama harinya.

Contoh : Input : 1 Output "Senin"
Input : 6 Output "Sabtu"

```
Program NamaHari
{ Mengisi TabelNamaHari yang akan memungkinkan untuk menuliskan nama hari:
tabulasi eksplisit nama hari berdasarkan indeks HariKe... }
```

```
KAMUS
   TabelNamaHari : array [1..7] of string
   procedure IsiTabHari     { mengisi tabel nama hari }
   HariKe : integer [1..7]  { nomor dari hari}
```

```
ALGORITMA
   IsiTabHari
   { Contoh pemanfaatan setelah Tabel TabNamaHari terdefinisi isinya }
   input (HariKe)
   output (TabelNamaHari_HariKe)
```

```
procedure IsiTabHari
{ Mengisi tabel nama hari }
{ I.S. : TabNamaHari tak terdefinisi }
{ F.S. : TabNamaHari siap dipakai, semua elemennya [1..7] sudah diisi }
```

```
KAMUS LOKAL
```

```
ALGORITMA
   TabelNamaHari_1 ← "Senin"
   TabelNamaHari_2 ← "Selasa"
   TabelNamaHari_3 ← "Rabu"
   TabelNamaHari_4 ← "Kamis"
   TabelNamaHari_5 ← "Jumat"
   TabelNamaHari_6 ← "Sabtu"
   TabelNamaHari_7 ← "Minggu"
```

Kasus 2: TABEL KATA

Didefinisikan bahwa kata adalah sebuah type yang menyimpan kata dan panjang katanya. Panjang maksimum sebuah kata adalah 50.

Maka dibuat kamus sebagai berikut:

```
KAMUS
   constant MaxKata : integer = 50
   type Kata : < TabKata : array[1..MaxKata] of character { definisi kata }
                 Length  : integer >                      { panjang kata }
```

Berikut ini adalah prosedur untuk membaca sebuah kata dari *keyboard*, dan menuliskan kata yang dibaca serta panjangnya ke layar dengan menggunakan kamus umum di atas:

```
procedure BacaTulisKata
{ I.S. : Sembarang }
{ F.S. : Sebuah kata dibaca dan dituliskan kembali di layar. Jika karakter
yang diketikkan melebihi panjang kata maksimum, pembacaan dihentikan dan
pada akhir proses dituliskan pesan. Jika program terus menerus membaca tanpa
menghentikan pembacaan ketika ukuran array dilampaui, program akan "abort".
Maka jika pengguna mengetikkan sejumlah karakter yang melebihi kapasitas
pendefinisian tabel TabKata, pembacaan dihentikan. }
```

```
KAMUS LOKAL
   K : Kata
   cc : character
   i  : integer
```

```
ALGORITMA
    { Baca kata huruf demi huruf, pembacaan diakhiri dengan huruf '@' }
    { Skema pemrosesan sekuensial dengan mark, dengan kasus kosong }
    input (cc)       { First-Elmt }
    K.Length ← 0     { Inisialisasi, harus di sini supaya kata "kosong"
                       terdefinisi }
    if (cc = '@') then
       output ("Tidak ada kata yang dibaca")   { Proses-Kasus-Kosong }
    else
       repeat
           K.Length ← K.Length + 1
           K.TabKata_K.Length ← cc             { Proses-Current-Elmt }
           input (cc)                          { Next-Elmt }
       until (cc = '@') or (K.Length = MaxKata)
       { Terminasi }
       depend on (cc)
           cc ≠ '@' : output ("Pembacaan kata dihentikan")
           cc = '@' : output ("Kata yang dibaca adalah : ")
       { Penulisan kata }
       i traversal [1..K.Length]
           output (K.TabKata_i)
       output ("Panjang kata : ", K.Length)
```

Latihan soal

1. Tuliskanlah kembali **prosedur IsiTabHari** pada contoh **program NamHari** dengan parameter formal nama Tabel yang menyimpan nama hari. Tuliskanlah pula **program NamaHari** sesudah **prosedur IsiTabHari** mempunyai parameter. Bandingkanlah ke dua solusi ini.

2. Buatlah kembali program untuk menghitung NEXTDAY (pada bagian FUNGSI) dengan mendefinisikan tabel jumlah hari per bulan dalam setahun. Berapa jumlah hari yang anda tentukan untuk bulan Februari? Bagaimana melakukan koreksi untuk tahun kabisat?

3. Jumlah hari dalam minggu dan jumlah bulan dalam setahun sudah tertentu. Bagaimana jika anda harus mendefinisikan tabel untuk jumlah elemen yang bisa bervariasi? Misalnya tabel yang setiap elemennya mewakili data murid dalam kelas, sedangkan jumlah murid per kelas bervariasi antara 20 -100 orang?

4. Buatlah sebuah unit program yang berisi:
   - pendefinisian suatu tabel bertype tertentu.
   - semua prosedur dan fungsi untuk memanipulasi tabel.

5. Andaikata pendefinisian kata dibuat demikian:

```
KAMUS
   constant MaxKata : integer = 50
   type Kata : array [1..MaxKata] of character    { Definisi kata }
```

   Maka:
   Kata hanya akan disimpan definisinya sebab panjang kata dapat "dikalkulasi" dari kata yang disimpan.
   Karakter terakhir kata dapat dikenali yaitu
   - karakter pada tabel Kata dengan indeks i, i = MaxKata, akhir tabel (jika tabel penuh), atau
   - karakter pada indeks tabel ke-i, dengan i < MaxKata dan $Kata_{i+1}$ = Blank,

   Tuliskanlah prosedur membaca dan menulis kata seperti pada kasus 2 untuk kamus ini.

Pemrosesan Sekuensial terhadap Tabel

Pemrosesan sekuensial terhadap tabel adalah pemrosesan sekuensial tanpa mark. Tabel memungkinkan adanya akses langsung jika indeks terdefinisi; karena indeks pada tabel mempunyai keterurutan tertentu, maka akses dari satu elemen ke elemen berikutnya dapat dilakukan dengan memanfaatkan keterurutan indeks tsb. Yang dimaksud dengan First-Elmt adalah elemen tabel dengan indeks terkecil, Next-Elmt dicapai melalui suksesor indeks. Model akses adalah model akses sekuensial tanpa mark, dan kondisi berhenti adalah jika indeks sudah mencapai harga indeks yang terbesar yang telah terdefinisi. Tabel tidak mungkin "kosong", artinya jika kita mendefinisikan tabel, maka minimal mengandung sebuah elemen.

```
KAMUS UMUM PEMROSESAN TABEL
    constant Nmin : integer = 1
    constant Nmax : integer = 100 { Nmin dan Nmax :
                                    batas bawah dan atas yg ditentukan}
    type ElType   : ... { suatu type terdefinisi, misalnya integer }
    i : integer [Nmin..Nmax]
    T : array [Nmin..Nmax] of ElType { tabel yang didefinisikan atas indeks
                                       i, dengan elemen bertype ElType }
    procedure Inisialisasi
    { Persiapan yang harus dilakukan sebelum pemrosesan }
    procedure Proses (input X : integer)
    { Proses terhadap Current-Elmt tabel T }
    procedure Terminasi
    { "Penutupan" yang harus dilakukan setelah pemrosesan selesai }
```

```
SKEMA PEMROSESAN TABEL T untuk indeks [Nmin..Nmax]
{ Traversal Tabel T untuk Indeks bernilai Nmin..Nmax }
```

```
SKEMA
    Inisialisasi
    i ← Nmin          { First-Elmt }
    iterate
       Proses(T_i)
    stop (i = Nmax)   { EOP }
       i ← i + 1      { Next-Elmt }
    Terminasi
```

Karena pemrosesan sekuensial terhadap semua elemen tabel sering dipakai, maka skema di atas dipersingkat cara penulisannya sebagai berikut:

```
SKEMA PEMROSESAN TABEL T untuk indeks [Nmin..Nmax]
{ Traversal Tabel T untuk Indeks bernilai Nmin..Nmax }
```

```
SKEMA
    Inisialisasi
    i traversal [NMin..NMax]
       Proses(T_i)
    Terminasi
```

Berikut ini diberikan beberapa contoh pemakaian skema pemrosesan tabel:

Contoh 1: Mengisi Tabel

```
Program ISITABEL1
{ Traversal untuk mengisi, dengan mebaca  nilai setiap elemen tabel dari
keyboard jika banyaknya elemen tabel yaitu N diketahui. Nilai yang dibaca
akan disimpan di T(NMin) s.d. T(N). Nilai N harus berada dalam daerah nilai
indeks yang valid }

KAMUS
    constant Nmin : integer = 1   { Nmin : batas bawah indeks }
    constant Nmax : integer = 100 { Nmax : batas atas indeks}
    i : integer [Nmin..Nmax]
    T : array [NMin..Nmax] of integer
    N : integer

ALGORITMA
    { Inisialisasi }
    repeat
        input (N)
    until (Nmin ≤ N ≤ Nmax)
    i traversal [NMin..N]
        input (T_i)
```

Contoh 2: Mengisi Tabel

```
Program ISITABEL1
{ Traversal untuk mengisi, dengan mebaca  nilai setiap elemen tabel dari
keyboard  yang diakhiri dengan 9999. Nilai yang dibaca akan disimpan di
T(Nmin) s.d. T(N).
Nilai N harus berada dalam daerah nilai indeks yang valid }

KAMUS
    constant Nmin : integer = 1
    constant NMax : integer = 100 { Nmin dan Nmax :
                                    batas bawah dan atas yg ditentukan }
    i : integer
    x : integer   { nilai yang dibaca dan akan disimpan sebagai elemen tabel }
    T : array [NMin..Nmax] of integer

ALGORITMA
    i ← NMin          { Inisialisasi }
    input (x)         { First-Elmt }
    while (x ≠ 9999) and (i ≤ Nmax) do
        T_i ← x       { Proses }
        i ← i + 1
        input (x)     { Next-Elmt }
    { x = 9999 or i > Nmax }
    if (i > Nmax) then
        output ("Tabel sudah penuh")
```

Contoh 3:
Menuliskan isi tabel secara "mundur" (mulai dari elemen dengan indeks yang terbesar yang terakhir sampai dengan yang pertama).

```
Program TULISTABELMundur
{ Menuliskan isi tabel dari indkes terbesar ke indeks terkecil }

KAMUS
   constant NMin : integer = 1
   constant NMax : integer = 100
   TabelInt : array [Nmin..Nmax] of integer
   i : integer [Nmin..Nmax]     { Nmin dan Nmax adalah indeks tabel }
   N : integer                  { ukuran efektif tabel yang terisi 1..N }

ALGORITMA
{ Di titik ini: tabel T [1..N] sudah diisi, algoritma berikut hanya
menuliskan mundur }
   i traversal [N..1]
      output (T_i)
```

Table Look Up, Searching

Dalam kehidupan sehari-hari, *"table look up"* (mencari suatu nilai dalam tabel) sering dipakai, misalnya dalam membaca daftar acara televisi, jadwal kereta api, melihat isi buku petunjuk telpon, dan lain-lain.

Dalam pemrosesan dengan komputer, proses ini penting karena sering dilakukan terhadap sekumpulan data yang disimpan dalam tabel. Ada beberapa variasi pencarian. Metoda mana yang dipakai menentukan kecepatan pencarian. Performansi pencarian secara lebih eksak akan dipelajari pada analisis algoritma.

Untuk semua persoalan pencarian pada sub bab ini, dipakai kamus umum sebagai berikut :

```
KAMUS UMUM
   constant  Nmax : integer = 100
   type TabInt : array [1..Nmax] of integer

{ Jika diperlukan sebuah tabel, maka akan dibuat deklarasi sebagai berikut }
   T : TabInt    { tabel integer }
   N : integer   { indeks efektif, 1 ≤ N ≤ Nmax+1 }
```

Pencarian Berturutan (*Sequential Search*)

Persoalan :
Diketahui sebuah tabel berisi harga integer T [1..N], yang telah diisi. Tuliskanlah algoritma yang jika diberikan sebuah X bernilai integer akan mencari apakah harga X ada dalam T secara sekuensial (berturutan) mulai dari elemen pertama sampai ketemu atau sampai elemen terakhir. Algoritma akan menghasilkan harga indeks IX dimana X diketemukan pertama kalinya, IX diberi harga 0 jika pencarian tidak ketemu.. Pencarian segera dihentikan begitu harga pertama diketemukan.

Contoh 1: N = 8, T berisi: { 1, 3, 5, -8, 12, 90, 3, 5} X = 5
Pemeriksaan dilakukan terhadap {1, 3, 5}
Output: IX = 3
Contoh 2: N = 4, T berisi: {11, 3, 5, 8} X = 100
Pemeriksaan dilakukan terhadap {11, 3, 5, 8}
Output: IX = 0

Berikut ini diberikan:
- dua buah versi algoritma pencarian yang benar, dan
- sebuah versi pencarian yang "salah", yang sering ditulis oleh pemula tanpa menyadari bahwa versi tersebut sebenarnya salah secara konseptual (algoritmik), walaupun mungkin saja akan berjalan dengan baik karena implementasi kompiler.

Algoritma pencarian secara sekuensial versi-1 (benar):
Skema pencarian tanpa boolean

```
procedure SEQSearchX1 (input T : TabInt, input N : integer,
                       output IX : integer, input X : integer)
{ Mencari harga X dalam Tabel T [1..N] secara sekuensial mulai dari T1.
  Hasilnya adalah indeks IX di mana T_ix = X (i terkecil) }
{ IX = 0 jika tidak ketemu. }
```

```
KAMUS LOKAL
   i : integer [1..Nmax]    { indeks untuk pencarian }
```

```
ALGORITMA
   i ← 1
   while (i < N) and (T_i ≠ X) do
       i ← i + 1
   { i = N or T_i = X }
   if (T_i = X) then
       IX ← i
   else   { T_i ≠ X }
       IX ← 0
```

Catatan :
1. Pada versi ini, elemen tabel yang terakhir diperiksa secara khusus.
2. Proses pada badan pengulangan hanya berisi "maju". Pencarian dihentikan karena ketemu, atau karena sudah tidak dapat "maju". Yang menentukan apakah pencarian ketemu atau tidak adalah kesamaan nilai elemen tabel ketika pencarian dihentikan.
3. Pada versi ini tidak diperlukan nama boolean seperti pada versi-2, namun jika dikehendaki bahwa hasilnya adalah sebuah nilai boolean (ketemu atau tidak ketemu), maka dapat dituliskan versi-1a sebagai berikut:

```
procedure SEQSearchX1a (input T : TabInt, input N : integer,
                        output Found : boolean)
{ Mencari harga X dalam Tabel T [1..N] secara sekuensial mulai dari T1
Hasilnya adalah sebuah nilai boolean Found (true jika ketemu, false jika
tidak. Pada versi ini informasi yang diperlukan adalah ketemu atau tidak
ketemu, namun kita tidak mendapatkan informasi di mana indeks nilai yang
dicari diketemukan }
```

```
KAMUS LOKAL
   i : integer [1..N]  { indeks untuk pencarian }
```

```
ALGORITMA
   i ← 1
   while (i < N) and (T_i ≠ X) do
      i ← i + 1
   { i = N or T_i = X }
   Found ← (T_i = X)
```

Skema Search dengan Boolean

Algoritma pencarian secara sekuensial versi-2 (benar):

```
procedure SEQSearchX2 (input T : TabInt, input N : integer,
                       input X : integer, output IX : integer,
                       output Found : boolean )
{ Mencari harga X dalam Tabel T[1..N] secara sekuensial mulai dari T_1,
  Hasilnya adalah indeks IX di mana T_i=X (i terkecil),
  IX = 0 jika tidak ketemu dan sebuah boolean Found (true jika ketemu). }
```

```
KAMUS LOKAL
   i : integer [1..N+1]  { indeks untuk pencarian }
```

```
ALGORITMA
   Found ← false  { awal pencarian, belum ketemu }
   i ← 1
   while (i ≤ N) and (not Found) do
      if (T_i = X) then
         Found ← true
      else
         i ← i + 1
   { i > N or Found }
   if (Found) then
      IX ← i
   else
      IX ← 0
```

Catatan :

1. Pada versi ini, semua elemen tabel diperiksa dalam badan pengulangan dengan instruksi yang sama.
2. Nilai tabel yang diperiksa indeksnya adalah dalam definisi tabel tersebut.

Algoritma pencarian secara sekuensial versi-3 (salah) :

```
procedure SEQSearchX3 (input T : TabInt, input N : integer,
                       input X : integer, output IX : integer,
                       output Found : boolean)
{ Mencari harga X dalam Tabel T [1..N] secara sekuensial mulai dari T1.
  Hasilnya adalah indeks IX di mana Ti = X (i terkecil),
  IX = 0 jika tidak ketemu dan sebuah boolean Found (true jika ketemu). }
KAMUS LOKAL
   i : integer [1..Nmax+1]      { indeks untuk pencarian }
ALGORITMA
   i ← 1
   while (i ≠ N) and (Ti ≠ X) do
      i ← i + 1
   { i = N+1 or Ti ≠ X }
   if (i ≤ N) then
      Found ← true; IX ← i
   else { i > N }
      Found ← false; IX ← 0
```

Catatan:

1. Pada versi-3 ini, semua elemen tabel diperiksa dalam badan pengulangan dengan instruksi yang sama.
2. Algoritma ini **salah** karena ada nilai tabel dengan indeks TIDAK TERDEFINISI yang diperiksa pada instruksi pengetesan kondisi *while* tabel tersebut. Beberapa pemroses bahasa akan melakukan optimasi dalam melakukan evaluasi ekspresi boolean "*and*". Jika operan pertama salah, maka pemroses bahasa tidak melakukan evaluasi lebih lanjut terhadap operan kedua. Jika optimasi ini dilakukan, maka program akan "selamat" (benar), jika tidak ada optimasi maka akan terjadi *index out of bound*. Namun operator semacam ini dalam evaluasi *and* dengan optimasi semacam ini bukan disebut sebagai operator boolean *and*, melainkan *and then*. Jadi, secara konseptual algoritmik, versi ini salah.

Sequential Search pada Tabel Terurut

Persoalan:

Diketahui sebuah tabel bilangan integer T [1..N], yang telah diisi, dan isinya terurut membesar (untuk setiap i ≤ [1..N-1], $T_i \le T_{(i+1)}$). Tuliskanlah algoritma, yang jika diberikan sebuah X bernilai integer akan mencari apakah harga X ada dalam T secara sekuensial mulai dari elemen pertama sampai ketemu atau sampai elemen terakhir. Prosedur akan menghasilkan harga indeks IX di mana X diketemukan pertama kalinya, IX diberi harga 0 jika pencarian tidak ketemu. Pencarian segera dihentikan begitu harga pertama diketemukan. Dengan memanfaatkan keterurutan, kondisi berhenti bisa dibuat lebih efisien.

Contoh:

N = 8, T berisi: { 1, 3, 5, 8, 12, 90, 311, 500} X = 5
Pemeriksaan dilakukan terhadap {1, 3, 5}, Output: IX = 3
N = 7, T berisi: { 11, 30, 50, 83, 99,123, 456} X = 100
Pemeriksaan dilakukan terhadap {11, 30, 50, 83, 99, 123}, Output: IX = 0

```
procedure SEQSearchSorted (input T : TabInt, input N : integer,
                           input X : integer, output IX : integer)
{ Mencari harga X dalam Tabel T [1..N] secara sekuensial mulai dari T1 }
{ Hasilnya adalah indeks IX di mana TIX = X; IX = 0 jika tidak ketemu }

KAMUS LOKAL
   i : integer [1..Nmax]    { indeks untuk pencarian }

ALGORITMA
   i ← 1
   while (i < N) and (Ti < X) do
      i ← i + 1
   { i = N or Ti ≥ X }
   if (Ti = X) then
      IX ← i
   else   { Ti ≠ X ➔ Ti > X }
      IX ← 0
```

Catatan :

1. Algoritma tersebut adalah modifikasi dari algoritma pencarian versi-1. Tentunya dapat pula dituliskan algoritma berdasarkan versi-2 yang memakai nilai boolean. Tuliskanlah sebagai latihan.
2. Algoritma tergantung kepada keterurutan nilai pada tabel: membesar atau mengecil.

Sequential Search Dengan Sentinel

Persoalan:

Diketahui sebuah tabel bilangan integer T [1..N], yang telah diisi. Tuliskanlah sebuah algoritma, yang jika diberikan sebuah X bernilai integer akan mencari apakah harga X ada dalam T secara sekuensial mulai dari elemen pertama tabel sampai ketemu atau sampai kepada elemen sentinel (berarti tidak ketemu). Prosedur akan menghasilkan harga indeks IX dimana X diketemukan pertama kalinya, IX diberi harga 0 jika pencarian tidak berhasil. Pencarian segera dihentikan ketika harga pertama diketemukan.

Dengan teknik sentinel, sengaja dipasang suatu elemen fiktif setelah elemen terakhir tabel, yang disebut SENTINEL. Elemen fiktif ini harganya adalah sama dengan elemen yang dicari. Akibatnya, pencarian akan selalu menghasilkan harga yang dicari; tetapi harus diperiksa lagi apakah posisi ketemu :

- di antara elemen tabel yang sebenarnya, atau
- sesudah elemen terakhir (berarti tidak ketemu, karena elemen fiktif)

Sentinel dapat diletakkan sebelum elemen pertama tabel atau sesudah elemen terakhir tabel tergantung kepada arah pencarian. Jika pencarian dilakukan secara “maju” (dari indeks terkecil sampai ke yang terbesar), maka sentinel diletakkan sesudah elemen terakhir tabel.

Teknik sentinel sangat efisien, terutama jika pencarian dilakukan sebelum penyisipan sebuah elemen yang belum terdapat di dalam tabel. Dengan teknik ini, jika tidak ketemu, sekaligus harga tersebut sudah ditambahkan. Hanya saja pemrogram harus

hati-hati dengan pendefinisian *range* dari indeks tabel. Pada kesempatan lain, akan ditunjukkan kasus dimana teknik ini sangat layak digunakan.
Contoh :

N = 8, T berisi: {1, 3, 5, 8, -12, 90, 3, 5} X = 5
T dijadikan {1, 3, 5, 8, 12, 90, 3, 5, 5}
Pemeriksaan dilakukan terhadap {1,3,5} Output : IX = 3
N = 4, T berisi: {11, 3, 5, 8} X = 100
Akibatnya minimal ukuran array harus 5, berarti N dijadikan 5
T dijadikan {11, 3, 5, 8, 100}
Pemeriksaan dilakukan terhadap {11, 3, 5, 8, 100}
Output: IX = 0

Kamus umum untuk tabel dengan sentinel harus mengandung sebuah elemen tambahan:

```
KAMUS UMUM
   constant  Nmax : integer  = 100
   type TabInt : array [1..Nmax+1] of integer
   N  : integer    { indeks efektif, maksimum tabel yang terdefinisi,
                     1 ≤ N ≤ Nmax+1 }

{ Jika diperlukan sebuah tabel, maka akan dibuat deklarasi sebagai berikut }
   T : TabInt      { tabel integer }
   N : integer     { indeks, 1 ≤ N ≤ Nmax+1 }
```

```
procedure SEQSearchWithSentinel (input T : TabInt, input N : integer,
                                 input X : integer, output IX : integer)
{ Mencari harga X dalam Tabel T [1..N] secara sekuensial mulai dari T1 }
{ Hasilnya adalah indeks IX di mana TIX = X (IX terkecil) }
{ IX = 0 jika tidak ketemu. Sentinel diletakkan di TN+1 }
```

```
KAMUS LOKAL
   i : integer [1..N+1] { indeks untuk pencarian }
```

```
ALGORITMA
   TN+1 ← X     { pasang sentinel }
   i ← 1
   while (Ti ≠ X) do  { Tidak perlu tes terhadap batas i,
                        karena pasti berhenti}
      i ← i + 1
   { T 1 = X; harus diperiksa apakah ketemunya di sentinel }

   if (i < N+1) then
       IX ← i         { ketemu pada elemen tabel }
   else   { i = N+1 }
      IX ← 0          { sentinel, berarti tidak ketemu }
```

Binary Search

Persoalan:
Diketahui sebuah tabel bilangan integer T [1..N], yang telah diisi dan isinya **terurut menaik**. Tuliskanlah sebuah prosedur BINSEARCHX, yang jika diberikan sebuah X bernilai integer akan mencari apakah harga X ada dalam T.secara dikotomik, yaitu setiap saat pemeriksaan dilakukan dengan mereduksi elemen tabel, yaitu terhadap separuh dari elemen tabel:

- bandingkan harga yang dicari dengan harga elemen tengah,
- jika sama, berarti ketemu,
- jika yang dicari lebih kecil, berarti pencarian dengan cara yang sama harus dilakukan terhadap elemen-elemen pada belahan atas,
- jika harga yang dicari lebih besar, berarti pencarian dengan cara yang sama harus dilakukan terhadap elemen-elemen pada belahan bawah.

Pencarian dengan cara ini akan mengurangi waktu pencarian karena pembandingan harga direduksi secara logaritmik (dengan basis 2). Kecepatan pencarian sebanding dengan ln (N). Namun cara pencarian ini hanya dapat dilakukan jika elemen tabel sudah terurut. Cara ini tidak mungkin dilakukan jika elemen tabel tidak terurut.

Algoritma pencarian secara dikotomik berikut akan menghasilkan sebuah harga boolean FOUND, yang bernilai *true* jika X ada dalam tabel, *false* jika tidak. Selain itu, harga indeks IX di mana X diketemukan pertama kalinya juga disimpan. Pencarian segera dihentikan begitu harga pertama diketemukan
Contoh:

N = 8, T berisi: {1, 3, 5, 8, 12, 90, 113, 500} X = 5
Pemeriksaan dilakukan terhadap {8,3,5}, Output: Found = true, IX = 3
N = 5, T berisi: {11, 37, 51, 80,90} X = 100
Pemeriksaan dilakukan terhadap {51,80,90}, Output: Found = false, IX = 0

Solusi-1: Versi dengan boolean

```
procedure BinSearch1 (input T : TabInt, input N : integer,
                      input X : integer, output Found : boolean)
{ Mencari harga X dalam Tabel T [1..N] secara dikhotomik }
{ Hasilnya adalah sebuah boolean Found, true jika ketemu }
{ Nilai elemen tabel terurut membesar: T1 ≤ T2 ≤ T3 ≤ ... ≤ TN }
```

```
KAMUS LOKAL
   Atas, Bawah, Tengah : integer { indeks atas, bawah, tengah:
                                   batas pemeriksaan }
```

```
ALGORITMA
   Atas ← 1; Bawah ← N     { Batas atas dan bawah seluruh tabel }
   Found ← false           { Mula-mula belum ketemu }
   while (Atas ≤ Bawah) and (not Found) do
      Tengah ← (Atas + Bawah) div 2
      depend on (T, Tengah, X)
         X = TTengah : Found ← true; IX ← Tengah
         X < TTengah : Bawah ← Tengah - 1
         X > TTengah : Atas ← Tengah + 1
   { Atas > Bawah or Found, harga Found menentukan hasil pencarian }
```

Catatan :

1. Semua pemeriksaan dilakukan dengan cara yang sama di dalam badan pengulangan.
2. Algoritma tersebut berlaku untuk elemen tabel yang terurut membesar, dan harus dimodifikasi untuk elemen tabel yang terurut mengecil. Tuliskanlah sebagai latihan.

Solusi-2: Versi tanpa boolean

```
procedure BinSearch1 (input T : TabInt, input N : integer,
                      input X : integer, output Found : boolean,
                      output IX : integer)
{ Mencari harga X dalam Tabel T [1..N] secara dikotomik }
{ Hasilnya adalah sebuah boolean Found, true jika ketemu dan IX yang
  merupakan indeks di mana X ditemukan }
{ Nilai elemen tabel terurut membesar:  T1 ≤ T2 ≤ T3  ...    ≤ TN }
```

```
KAMUS LOKAL
   Atas, Bawah, Tengah : integer { indeks atas, bawah, tengah:
                                   batas pemeriksaan }
```

```
ALGORITMA
   Atas ← 1; Bawah ← N         { Batas atas dan bawah seluruh tabel }
   Tengah ← (Atas + Bawah) div 2
   while (Atas < Bawah) and (X ≠ T_Tengah) do
      depend on (T, Tengah, X)
         X < T_Tengah : Bawah ← Tengah - 1
         X > T_Tengah : Atas ← Tengah + 1
      Tengah ← (Atas + Bawah) div 2
   { Atas ≥ Bawah or X = T_Tengah }
   if (X = T_Tengah) then
       Found ← true; IX ← Tengah
   else { X ≠ T_Tengah tidak ketemu }
        Found ← false; IX ← 0
```

Catatan:

Pemeriksaan elemen terakhir dilakukan secara khusus di luar badan pengulangan. Perhatikan bagaimana indeks Tengah didefinisikan.

Harga Ekstrem Tabel

Selain pencarian suatu harga dalam tabel, salah satu proses lain yang penting, adalah mencari harga ekstrem (maksimum dan minimum). Contoh dalam kehidupan sehari-hari: mencari juara kelas, mencari data percobaan yang merupakan harga ekstrem, dan lain-lain. Berikut ini akan diberikan algoritma untuk mencari harga maksimum dari tabel:

Persoalan:
Diketahui sebuah tabel bilangan integer T [1..N], yang telah diisi. Tuliskanlah sebuah program Max, yang menghasilkan harga maksimum dari elemen tabel: $i \in [1..N]$, $T_i \leq Max$.

Contoh:

N = 8, T berisi: {1, -3, 5, 8, -12, 90, 3, 5}
Output: Maximum adalah 90
N = 11, T berisi: {-11, 3, 45, 8, 3, 45, -6, 7, 8, 9, 1}
Output: Maksimum adalah 45

Algoritma maksimum versi-1:

```
procedure MAX1 (input T : TabInt, input N : integer, output MAX : integer)
{ Pencarian harga Maksimum: }
{ I.S. Tabel tidak kosong, karena jika kosong maka harga maksimum tidak
terdefinisi, N >= 0 }
{ F.S. Menghasilkan harga maksimum MAX pada tabel T [1..N] secara sekuensial
mulai dari indeks 1..Nmax }
```

```
KAMUS LOKAL
    i : integer   { indeks untuk pencarian }
```

```
ALGORITMA
    MAX ← T1
    i ← 2
    while (i ≤ N) do
        if (MAX < Ti) then
            MAX ← Ti
        i ← i + 1
    { i > N : semua elemen sudah selesai diperiksa }
```

Catatan :

1. Pada algoritma maksimum versi-1 ini, elemen pertama tabel diproses secara khusus (bukan di dalam pengulangan).
2. Algoritma tersebut menghasilkan nilai maksimum, namun tidak diketahui posisi (indeks) di mana nilai maksimum tersebut berada.
3. Seringkali dalam suatu proses, kita membutuhkan indeks di mana nilai maksimum tersebut berada (terutama jika nilai maksimum muncul beberapa kali). Algoritmanya dituliskan pada versi-2.

Algoritma maksimum versi-2:

```
procedure MAX2 (input T : TabInt, input N : integer, output IMax : integer )
{ Pencarian indeks dengan harga Maksimum}
{ I.S. Tabel tidak kosong, karena jika kosong maka harga maksimum tidak
terdefinisi, N > 0 }
{ F.S. Menghasilkan indeks IMax terkecil, dengan harga T_Imax dalam tabel T
[1..N] adalah maksimum }
```

```
KAMUS LOKAL
    i : integer
```

```
ALGORITMA
    IMax ← 1
    i ← 2
    while (i ≤ N) do
        if (T_IMax < T_i) then
            IMAX ← i
        i ← i + 1
    { i > N : semua elemen sudah selesai diperiksa }
```

Catatan :

1. Pada algoritma maksimum versi-2 ini, elemen pertama tabel diproses secara khusus (bukan di dalam pengulangan).
2. Algoritma tersebut TIDAK menghasilkan nilai maksimum, namun karena indeks di mana nilai maksimum tersebut berada diketahui maka nilai maksimum dapat diakses.
3. Jika bekerja dengan tabel, lebih disukai bekerja dengan indeks karena berkat indeks nilai dari elemen dapat diakses secara langsung. Sebaliknya, hanya mengetahui suatu nilai, tidak sederhana untuk mengetahui indeks dimana nilai tersebut berada, yaitu harus dilakukan dengan proses pencarian.

Algoritma maksimum versi-3:

Diketahui sebuah tabel bilangan integer T [1..N], yang telah diisi dengan bilangan integer positif atau sama dengan nol. Tuliskanlah sebuah prosedur MAXPOS, yang menghasilkan harga maksimum dari elemen tabel.

Contoh:

N = 8, T berisi: {1, 3, 5, 8, 12, 90,0, 3, 5}
Output: Maksimum adalah 90
N = 11, T berisi: {11, 3, 45, 8, 3, 45, 6, 7, 8, 9, 1}
Output: Maksimum adalah 45

```
procedure MAXPOS (input T: TabInt, input N : integer,
                  output MAX : integer )
{ Pencarian harga Maksimum di antara bilangan positif }
{ I.S. Tabel mungkin kosong: N=0, semua elemen tabel  T bernilai positif }
{ F.S. Max berisi nilai elemen tabel yang maksimum }
{ Menghasilkan harga Maksimum (MAX) tabel T [1..N] secara sekuensial mulai
dari T1 }
```

```
KAMUS LOKAL
   i : integer
```

```
ALGORITMA - 1
   MAX ← -9999  { inisialisasi dengan nilai yang pasti dapat digantikan!
                  dalam hal ini nilai minimum representasi integer }
   i traversal [1..N]
      if (MAX < Ti) then
         MAX ← Ti
   { i = ??, semua elemen sudah selesai diperiksa }
```

```
ALGORITMA - 2
   MAX ← -9999  { inisialisasi dengan nilai yang pasti dapat digantikan!
                  dalam hal ini nilai minimum representasi integer }
   i ← 1
   while (i ≤ N) do
      if (MAX < Ti) then
         MAX ← Ti
      i ← i + 1
   { i = N+1,  semua elemen sudah selesai diperiksa }
```

Catatan :

1. Semua elemen tabel diproses dengan cara yang sama, sehingga nilai maksimum sebelum elemen pertama diperiksa harus didefinisikan. Algoritma tersebut memakai suatu nilai yang didefinisikan oleh pemrogram di luar nilai tabel (max = -9999), yang pasti digantikan oleh salah satu elemen tabel. Algoritma ini hanya berlaku jika semua elemen tabel positif sesuai dengan spesifikasi. Nilai yang dipilih untuk inisialisasi nilai maksimum harus merupakan nilai yang tepat tergantung kondisi tabel.
2. Versi-versi sebelumnya lebih baik, bersifat umum dan tidak memakai suatu nilai di luar nilai tabel.

Internal Sorting

Sorting atau pengurutan data adalah proses yang sering harus dilakukan dalam pengolahan data. Bahkan mesin otomatik yang pertama kali lahir adalah mesin pengurut, dan masih dipakai sampai saat ini misalnya untuk menyortir surat berkode pos di kantor pos dengan mesin terotomatisasi. Dibedakan dua macam pengurutan :

- **pengurutan internal**, yaitu pengurutan terhadap sekumpulan data yang disimpan dalam media internal komputer yang dapat diakses setiap elemennya secara langsung. Maka dapat dikatakan sebagai pengurutan tabel.
- **pengurutan eksternal**, yaitu pengurutan data yang disimpan dalam memori sekunder, biasanya data bervolume besar sehingga tidak mampu untuk dimuat semuanya dalam memori.

Algoritma pengurutan adalah salah satu contoh solusi algoritmik yang kaya: satu macam persoalan (pengurutan), dapat dilakukan dengan puluhan macam algoritma. Algoritma pengurutan internal yang utama antara lain:

- Counting Sort
- Maximum Sort
- Insertion Sort
- Bubble sort
- Shaker sort
- Heap Sort
- Shell sort
- Quick Sort
- Radix sort
- ...

Berikut ini hanya akan dibahas pengurutan internal, itupun hanya 4 metoda yang pertama, yang paling sederhana.Metoda pengurutan lain lebih canggih (ditinjau dari analisis maupun dari struktur datanya tidak dibahas di sini, akan dipelajari pada mata kuliah yang lain). Algoritmanya dapat dipelajari dari buku-buku [Knuth-73], [Wirth-86]. Buku [Knuth-73] adalah sebuah buku yang khusus membahas tentang "*Sorting* and *Searching*", setiap algoritma dan performansinya dibahas lebih mendalam. Buku [Wirth-86] sifatnya lebih metodologis dan pedagogis, versi yang ditampilkan bukan versi yang optimum.

Performansi pengurutan data sangat menentukan performansi sistem, karena itu pemilihan metoda pengurutan yang cocok akan berperan dalam.suatu aplikasi. Biasanya selain ditentukan peformansi rata-rata, juga ditentukan performansi terjelek dan performansi terbagus. Untuk beberapa aplikasi, performansi yang "stabil" (tidak terlalu mencolok bedanya antara kasus terjelek dan terbagus) perlu juga dipertimbangkan.

Untuk semua persoalan *sorting*, dipakai kamus sebagai berikut:

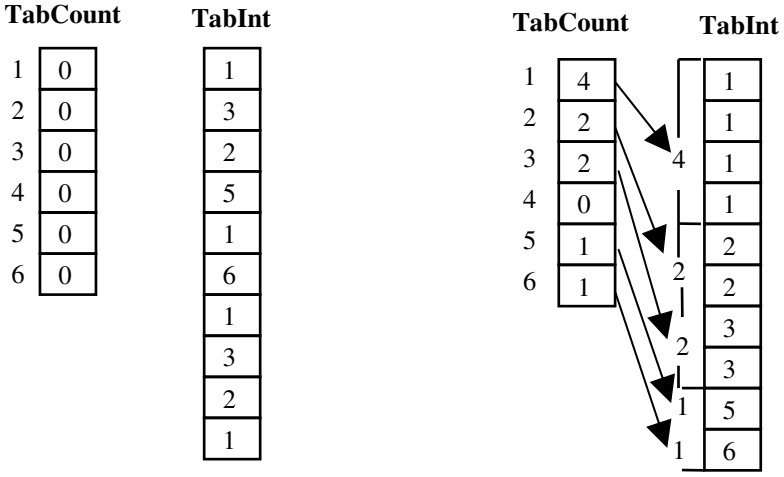
```
KAMUS
    constant Nmax : integer = 100
    type TabInt : array [1..Nmax] of integer
    N : integer   { indeks efektif, maksimum tabel yang terdefinisi,
                    N < Nmax }

{ Jika diperlukan sebuah tabel, maka akan dibuat deklarasi sebagai berikut }
    T : TabInt    { tabel integer }
    N : integer   { indeks efektif, 1 ≤ N ≤ Nmax+1 }
```

Dan persoalannya adalah:
Diberikan sebuah tabel integer T [1..N] yang isinya sudah terdefinisi. Tuliskan sebuah algoritma yang mengurutkan elemen tabel sehingga terurut membesar:
$T_1 \leq T_2 \leq T_3 \ldots\ldots \leq T_N$

Pengurutan dengan Pencacahan

Pengurutan dengan pencacahan adalah pengurutan yang paling sederhana. Jika diketahui bahwa data yang akan diurut mempunyai daerah jelajah (*range*) tertentu, dan merupakan bilangan bulat, misalnya [Min..Max] maka cara paling sederhana untuk mengurut adalah :

1. Sediakan array TabCount [Min..Max] yang diinisialisasi dengan nol, dan pada akhir proses TabCount$_i$ berisi banyaknya data pada tabel asal yang bernilai i.
2. Tabel dibentuk kembali dengan menuliskan kembali harga-harga yang ada.



```
procedure CountSORT (input/output T : TabInt, input N : integer)
{ Mengurut tabel integer [1..N] dengan pencacahan }

KAMUS LOKAL
{ ValMin dan ValMax adalah batas minimum dan Maximum harga yg tersimpan
dalam T, harus diketahui }
   TabCount : array [ValMin..Valmax] of integer [0..NMax]
   i : integer { indeks untuk traversal tabel }
   K : integer { jumlah elemen T yang sudah diisi pada proses pembentukan
                 kembali }

ALGORITMA
   { Inisialisasi TabCount }
   i traversal [ValMin..ValMax]
      TabCount_i ← 0
   { Counting }
   i traversal [1..N]
      TabCount_T_i ← TabCount_T_i + 1
   { Pengisian kembali : T_1 ≤ T_2 ... ≤ T_N }
   K ← 0
   i traversal [ValMin..ValMax]
      if (TabCount_i ≠ 0) then
         repeat TabCount_i times
            K ← K + 1
            T_K ← i
```

Catatan :
$TabCount_{Ti_i}$ dituliskan untuk menunjukkan bahwa indeks T adalah i, dan T_i merupakan indeks dari TabCount.

Pengurutan Berdasarkan Seleksi

Contoh: Maksimum suksesif:

Idenya adalah menghasilkan nilai maksimum tabel (untuk efisiensi, hanya indeks dimana harga maksimum ditemukan yang dicatat), kemudian menukarnya dengan elemen terujung; elemen terujung ini "diisolasi" dan tidak diikut sertakan pada proses berikutnya. proses diulang untuk sisa tabel. Karena elemen terujung bernilai maksimum adalah indeks pertama tabel, maka tabel terurut **mengecil**:
$T_1 \geq T_2 \geq T_3 \geq ... \geq T_N$. Dengan demikian, proses dilakukan sebanyak N tahapan (yang dalam sorting disebut sebagai "pass"):

1 Tentukan IMAX [1..N], T_{Imax} adalah maksimum dari T[1..N]
  Tukar T_{Imax} dengan T_1
2 Tentukan IMAX [2..N], T_{Imax} adalah maksimum dari T[2..N]
  Tukar T_{Imax} dengan T_2
3 Tentukan IMAX [3..N], T_{Imax} adalah maksimum dari T[3..N]
  Tukar T_{Imax} dengan T_3

...

N-1 Tentukan IMAX [N-1..N], T_{Imax} adalah maksimum dari T[N-1..N]
  Tukar T_{Imax} dengan T_{N-1}
N T [1..N] sudah terurut: $T_1 \geq T_2 \geq T_3 \geq ... \geq T_N$

```
procedure MAXSORT (input/output T : TabInt, input N : integer)
{ Mengurut tabel integer [1..N] terurut mengecil dengan maksimum suksesif }

KAMUS LOKAL
   i : integer     { indeks untuk traversal tabel }
   Pass : integer  { tahapan pengurutan }
   Temp : integer  { memorisasi harga untuk penukaran }
   IMax : integer  { indeks, di mana T [1..pass] bernilai maksimum }

ALGORITMA
   Pass traversal [1..N-1]
      { Tentukan Maximum [Pass..N] }
      IMax ← pass
      i traversal [pass+1..N]
         if (T_IMax < T_i ) then
            IMax ← i
      { T_IMax adalah maximum T[pass..N] }
      { Tukar T_IMax dengan T_Pass  }
      Temp ← T_IMax
      T_IMax ← T_Pass
      T_Pass ← Temp
      { T [1..Pass] terurut: T_1 ≥ T_2 ≥ T_3 ≥ ... ≥ T_Pass }
   { Seluruh tabel terurut, T_1 ≥ T_2 ≥ T_3 ..... ≥ T_N }
```

Pengurutan Berdasarkan Penyisipan

Contoh: Insertion Sort

Idenya adalah mencari tempat yang "tepat" untuk setiap elemen tabel, dengan cara sequential search, kemudian setiap kali menyisipkan sebuah elemen tabel yang diproses ke tempatnya yang seharusnya. Proses dilakukan sebanyak N tahapan (yang dalam *sorting* disebut sebagai "pass"):

1 T_1 dianggap sudah tepat tempatnya

2 T_2 harus dicarikan tempat yang tepat pada T[1..2], yaitu i
Sisipkan T_2 pada T_i. T [1..2] terurut membesar

3 T_3 harus dicarikan tempat yang tepat pada T[1..3], yaitu i
Sisipkan T_3 pada T_3. T [1..3] terurut membesar

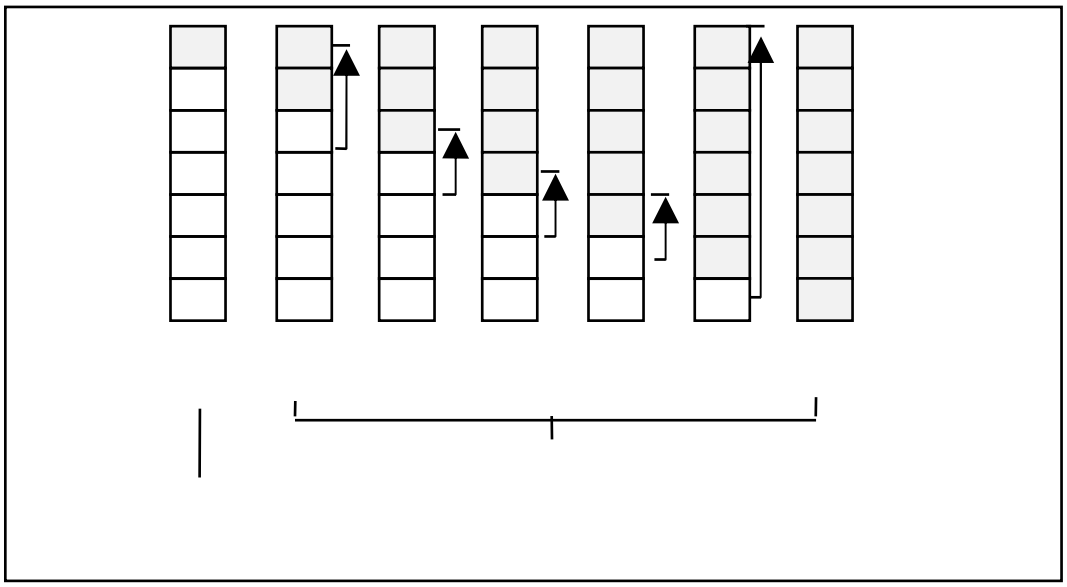
...

N-1 T_{N-1} harus dicarikan tempat yang tepat pada T[1..N-1], yaitu i
Sisipkan T_{N-1} pada T_i..T [1..N-1] terurut membesar

N T [1..N] sudah terurut : $T_1 \le T_2 \le T_3$ ..... $\le T_N$

Pada setiap Pass, tabel "terdiri dari" dua bagian : yang sudah terurut yaitu [1..Pass - 1] dan sisanya [Pass..Nmax] yang belum terurut. Ambil elemen T_{Pass}, sisipkan ke dalam T[1..Pass-1] dengan tetap menjaga keterurutan. Untuk menyisipkan T_{Pass} ke T_i, harus terjadi "pergeseran" elemen tabel T [i..Pass]. Pergeseran ini dapat dilakukan sekaligus dengan pencarian i. Pencarian dapat dihentikan segera dengan memanfaatkan sifat keterurutan T[1..Pass]. Metoda pengurutan ini cukup efisien ($\cong$ N) terutama untuk N yang "kecil".

Isi T pada Insertion Sort



```
procedure InsertionSORT (input/output T : TabInt, input N : integer)
{ Mengurut tabel integer [1..N] dengan insertion sort }

KAMUS LOKAL
   i : integer        { indeks untuk traversal tabel }
   Pass : integer     { tahapan pengurutan }
   Temp : integer

ALGORITMA
   { T1 adalah terurut}
   Pass traversal [2..N]
      Temp ← TPass          { Simpan  harga TPass
                              Supaya tidak tertimpa krn pergeseran  }
      { Sisipkan elemen ke Pass dalam T [1..Pass-1] sambil menggeser: }
      i ← Pass-1
      { Cari i, Temp < Ti and i > 1 }
      while (Temp < Ti) and (i > 1) do
         Ti+1 ← Ti          { Geser }
         i ← i - 1          { Berikutnya }
      { Temp ≥ Ti (tempat yg tepat) or i = 1 (sisipkan sbg. elmt. pertama) }
      depend on (T, i, Temp)
         Temp ≥ Ti    : Ti+1 ← Temp     { Menemukan tempat yg tepat }
         Temp < Ti    : Ti+1 ← Ti
                        Ti ← Temp       { sebagai elemen pertama }
      { T [1..Pass] terurut membesar: T1 ≤ T2 ≤ ... ≤ TPass }
   { Seluruh tabel terurut, karena Pass = N : T1 ≤ T2 ≤ T3 ≤ ... ≤ TN }
```

Pencarian dengan sentinel dapat dimanfaatkan pada Insertion Sort. Pada kasus ini, sentinel ditaruh pada T_0 karena pada tahapan ke-**Pass**, pencarian sekuensial dilakukan pada T [pass-1..1]. Berikut ini adalah algoritma untuk *insertion sort* dengan sentinel:

```
procedure InsertionSORTWithSentinel(input/output T: TabInt,
                                    input N : integer)
{ mengurut tabel integer [1..N] dgn insertion sort, pencarian dengan
sentinel}
```

```
KAMUS LOKAL
   i : integer      { indeks untuk traversal tabel }
   Pass : integer   { tahapan pengurutan }
   Temp : integer   { Menyimpan harga Tab_Pass spy tidak tertimpa krn
                      pergeseran }
```

```
ALGORITMA
   { T_1 adalah terurut }
   Pass traversal  [2..N]
      { Simpan dulu harga T(Pass) supaya tidak tertimpa karena pergeseran }
      Temp ← T_Pass
      T_0 ← Temp
      { Sisipkan elemen ke Pass dalam T[1..Pass-1] sambil menggeser }
      i ← Pass-1
      { Cari i, Temp < T_i and i > 1 }
      while (Temp < T_i) do
         T_i+1 ← T_i       { Geser}
         i ← i - 1         { Berikutnya }
      { Temp ≥ T_i }
      T_i+1 ← Temp         { Sisipkan }
      { T[1..Pass] terurut: T_1 ≤ T_2 ≤ T_3 ≤ ... ≤ T_Pass }
   { Seluruh tabel terurut, karena Pass = N : T_1 ≤ T_2 ≤ T_3 ≤ ... ≤ T_N }
```

Pengurutan Berdasarkan Petukaran Harga

Contoh: Bubble Sort

Idenya adalah gelembung air yang akan "mengapung" untuk tabel yang terurut membesar : elemen bernilai kecil akan "diapungkan" (ke indeks terkecil), artinya diangkat ke “atas” (indeks terkecil) melalui pertukaran. Proses dilakukan sebanyak N tahapan (yang dalam *sorting* disebut sebagai "pass").Pada setiap Pass, tabel "terdiri dari" dua bagian : bagian yang sudah terurut yaitu [1..Pass-1] dan ide dasarnya adalah mengapungkan elemen ke "pass" sampai pada tempatnya.

1 Untuk setiap dua elemen suksesif T_K dan T_{K-1}, K [2..N],
$T_{K-1} \le T_K$, dengan demikian T_K harus ditukar dengan T_{K-1} jika sifat di atas tidak dipenuhi.
Karena dilakukan secara sekuensial, T_1 berisi harga terkecil

2 Untuk setiap dua elemen suksesif T_K dan T_{K-1}, K [3..N],
$T_{K-1} \le T_K$, dengan demikian T_K harus ditukar dengan T_{K-1}
jika sifat di atas tidak dipenuhi
Karena dilakukan secara sekuensial, T[1..2] terurut

3 Untuk setiap dua elemen suksesif T_K dan T_{K-1}, K [4..N],
$T_{K-1} \le T_K$, dengan demikian T_K harus ditukar dengan T_{K-1} jika sifat di atas tidak dipenuhi.
Karena dilakukan secara sekuensial, T[1..3] terurut

...

N-1 Untuk setiap dua elemen suksesif T_K dan T_{K-1}, K [N-1..N],
$T_{K-1} < T_K$, dengan demikian T_K harus ditukar dengan T_{K-1}
jika sifat di atas tidak dipenuhi

Karena dilakukan secara sekuensial, T[1..N-1] terurut
T [1..N] sudah terurut : $T_1 \le T_2 \le T_3 \le ... \le T_N$

```
procedure BubbleSort (input/output T : TabInt, input N : integer)
{ Mengurut tabel integer [1..N] dengan bubble sort }

KAMUS LOKAL
   i, K : integer  { indeks untuk traversal tabel }
   Pass : integer  { tahapan pengurutan }
   Temp : integer  { Memorisasi untuk pertukaran harga }

ALGORITMA
   Pass traversal [1..N-1]
      K traversal [N..Pass+1]
         if (T_K < T_K-1) then
            Temp ← T_K
            T_K ← T_K-1
            T_K-1 ← Temp
   { T [1..Pass] terurut: T_1 ≤ T_2 ≤ T_3 ≤ ... ≤ T_Pass }
{ Seluruh tabel terurut, karena Pass = N: T_1 ≤ T_2 ≤ T_3 ≤ ... ≤ T_N }
```

Sebetulnya proses dapat dihentikan jika tidak terjadi lagi pertukaran. Manfaatkan sifat ini dengan memakai sebuah besaran boolean, dan tuliskanlah algoritmanya untuk memperoleh versi yang optimum.

Versi asli metoda ini biasanya paling diingat karena prinsipnya yang "alamiah", tapi performansinya tidak bagus (kecuali versi yang sudah dibuat efisien), maka tidak direkomendasikan untuk dipakai.

```
procedure BubleSortPlus (input/output T : TabInt, input N : integer)
{ Mengurut tabel integer [1..N] dengan bubble sort }
{ Pengurutan dihentikan jika tak ada pertukaran  lagi }
```

```
KAMUS LOKAL
   i : integer     { indeks untuk traversal tabel }
   Pass : integer  { tahapan pengurutan }
   Temp : integer  { Memorisasi untuk pertukaran harga }
   Tukar : boolean { true jika dalam satu pass ada pertukaran}
```

```
ALGORITMA
   Pass ←1
   Tukar ← true   { masih harus ada pertukaran }
   while (Pass ≤ N-1) and (Tukar) do
      Tukar ← false
      K traversal [N..Pass+1]
         if (T_K < T_K-1) then
            Temp ← T_K
            T_K ← T_K-1
            T_K-1 ← Temp
            Tukar ← true
      { T[1..Pass] terurut: T_1 ≤ T_2 ≤ T_3 ≤ ... ≤ T_Pass }
      { Tukar = true jika ada pertukaran }
      Pass ← Pass + 1   { ke pas yang berikutnya }
   { Seluruh tabel terurut, karena Pass = N: T_1 ≤ T_2 ≤ T_3 ≤ ... ≤ T_N }
```

Latihan Soal

Pemrosesan Sekuensial

Tuliskanlah algoritma yang masukannya adalah harga-harga integer antara 0 s/d 100 yang disimpan dalam tabel bilangan integer TABNILAI [1..100] dan melakukan proses sebagai berikut:

- menghitung nilai rata-rata dari harga yang disimpan dalam tabel tersebut.
- mengkonversi setiap angka ke dalam skala 'A' s/d 'E' dengan kriteria:
  ≥ 80 : 'A'
  ≥ 70 : 'B'
  ≥ 55 : 'C'
  ≥ 40 : 'D'
  < 40 : 'E'
  dan menyimpan hasil konversi setiap elemen tersebut ke dalam sebuah tabel lain.
- menghitung frekuensi relatif setiap nilai 'A', 'B', 'C', 'D' dan 'E' pada tabel yang dihasilkan.

Searching

1. Jika elemen dari tabel terurut, maka kondisi berhenti pada sequential search dengan sentinel juga dapat dibuat lebih efisien. Tuliskan algoritma Sequential search dengan sentinel, jika elemen tabel teurut mengecil, untuk setiap $i \in [1..N]$, dan $T_i \leq T_{i+1}$.

2. Diberikan sebuah tabel TABMK yang setiap elemennya berisi kode matakuliah (integer), dengan catatan bahwa maksimum matakuliah yang mungkin ada adalah 100. Banyaknya matakuliah yang sudah ada dalam tabel adalah NMK. Tuliskanlah sebuah algoritma yang menambahkan sebuah kode mata kuliah MKX, jika dan hanya jika MKX belum ada dalam tabel:
   - penambahan selalu dilakukan sebagai elemen terakhir tabel matakuliah.
   - elemen tabel TABMK [1..NMK] diurut membesar dan penambahan MKX harus tetap menjaga keterurutan kode matakuliah.

3. **Tabel Kode**
   Diberikan suatu tabel yang setiap elemennya ber-type bentukan :
   <kode : integer, deskripsi : string>
   Tabel terdefinisi dengan indeks [1..Nmax], namun isi efektif tabel adalah NELMT. Kode yang muncul pada tabel tersebut harus unik (hanya muncul satu kali saja). Tuliskanlah sebuah prosedur untuk melakukan penambahan kode baru dan deskripsinya, yang tetap membuat kode dalam tabel unik (artinya, sebelum melakukan penambahan elemn tabel, harus dilakukan pemeriksaan terlebih dulu). Pemeriksaan yang harus dilakukan adalah terhadap keunikan kode dan juga apakah masih ada "tempat kosong" dalam tabel. Implementasikan prosedur dengan cara pencarian dengan sentinel, sebab cara ini merupakan cara yang paling efisien untuk persoalan ini.

4. Jadwal Kereta Api

Diberikan suatu tabel yang indeknya terdefinisi untuk[1..N] yang setiap elemennya berisi nomor kereta api, kota berangkat, kota tujuan, jam berangkat, jam kedatangan dan tarif. Definisikan type yang cocok dan "masuk akal" untuk semua informasi tersebut, dan tuliskanlah algoritma yang menawarkan menu INFO-KA untuk menerima pilihan serta mengaktifkan prosedur yang sesuai untuk setiap pilihan. Pemakai boleh mengulang permintaan informasi sampai menekan Q (yang berarti keluar dari program.

Menu yang ditawarkan untuk pengguna adalah membuat daftar:

1 semua kereta api yang kota tujuannya adalah kota X dan biayanya
2 semua kereta api yang berangkat sebelum jam Y
3 semua kereta api dan jam berangkatnya, dari kota X menuju kota Y
4 kereta api pertama yang berangkat sebelum jam 9 pagi dari kota X
5 kereta api terakhir untuk menuju kota X dari kota Y
6 kereta termahal dan termurah untuk berangkat dari kota X menuju kota Y
7 kereta yang paling cepat sampai dari kota X ke kota Y
8 kereta api dari kota X menuju kota Y yang berangkat antara jam 7 pagi s/d 12 siang
9 kereta api yang tiba di kota Y dari kota Z sebelum jam 18.00
10 semua kereta api dari kota X untuk menuju kota Y, namun kereta langsung dari X ke Y mungkin tidak ada dalam elemen tabel (harus berhenti dulu di satu atau beberapa kota antara, untuk selanjutnya mengambil kereta api yang paling dulu berangkat dari kota antara itu untuk menuju ke kota antara lain atau kota tujuan)
11 semua kereta api yang ada untuk dibuat agen perjalanan.
Q Keluar dari Menu

Tuliskan algoritma lain yang menawarkan menu administrator dengan pilihan:

1 Menginisialisasi Tabel Jadwal Kereta Api
2 Menambahkan sebuah nomor kereta baru dan semua atributnya
3 Menghapus sebuah nomor kereta
4 Mengganti jadwal sebuah nomor kereta
5 Info kereta api (yang memanggil prosedur pada soal 10.1)
Q Keluar dari sistem

5. Ticketing

Tempat duduk pada suatu type pesawat penumpang dinomori dengan angka dan abjad 'A'..'J' (misalnya 1A, 10F, 3D, ...). Status tempat duduk disimpan sebagai suatu nilai boolean (true jika isi, false jika kosong) dalam suatu tabel dengan indeks [1..Nmax], dan tempat duduk yang tersedia adalah Nseat.

Buatlah kamus yang sesuai untuk pencarian tempat duduk kosong dalam pesawat, dan tuliskanlah prosedur-prosedur dengan deskripsi sebagai berikut:

- prosedur untuk mencari sebuah tempat duduk yang kosong dalam pesawat.
- prosedur untuk mencari N buah tempat duduk dalam pesawat yang kosong
- prosedur untuk mencari N buah tempat duduk ($2 \le N \le 4$) yang masih kosong dan berdampingan dalam pesawat.

Tempat duduk dalam pesawat adalah berdampingan jika bernomor :

A, B (untuk N = 2)
C, D (untuk N = 2)
E, F (untuk N = 2)
G, H, I, J (untuk N = 3 atau 4)

6. PASSWORD

Suatu jaringan komputer hanya mampu menangani maksimum 100 orang user. Informasi pengguna suatu jaringan komputer disimpan dalam suatu tabel, dengan informasi:

```
< User_id : string,
  Nama : string,
  Prioritas : integer,
  Password : array [1..20] of character >
```

User_id dalam tabel tersebut adalah unik, dan password disimpan dalam bentuk "rahasia", dengan rumus transformasi huruf yang diberikan dalam sebuah tabel, misalnya untuk beberapa huruf diberikan dalam tabel sebagai berikut :

Huruf yang diketikkan	Disimpan sebagai
A	Z
B	Y
C	X
D	W
E	V
F	U
G	T
1	%
2	#
@	$
#	@

Tuliskanlah prosedur-prosedur untuk pengelolaan user dan password, misalnya:

- menambahkan seorang user baru,
- menonaktifkan seorang user,
- mengubah password (harus konfirmasi dua kali sebelum mengubah),
- mengkoreksi nama yang sudah ada,
- memeriksa apakah user berhak memakai jaringan pada saat *log-in*.

Untuk setiap prosedur, buatlah spesifikasi yang jelas dan berikan justifikasi mengenai algoritma yang dipilih (misalnya metoda pencarian, ...)

7. BINARY SEARCH

Bacalah baik-baik algoritma di bawah ini, dan carilah kasus di mana algoritma tersebut salah.

```
procedure BinSearchX (input T : TabInt, input N : integer,
                      input X : integer, output IX : integer,
                      output Found : boolean )
{ Binary search }
{ Mencari harga X dalam Tabel T [1..N] secara dikotomik }
{ Hasilnya adalah indeks IX dimana T_i = X  IX = 0 jika tidak ketemu. }
{ Nilai elemen tabel terurut membesar:  T_1 ≤ X T_2 ≤ T_3 ≤ ... ≤ T_N }
```

```
KAMUS LOKAL
   i : integer [1..Nmax]          { indeks untuk pencarian }
   Atas, Bawah, Tengah : integer { indeks atas, bawah, tengah:
                                   batas pemeriksaan }
```

```
ALGORITMA
   Atas ← 1; Bawah ← N         { Batas atas dan bawah seluruh tabel }
   Found ← false; IX ← 0       { Mula-mula belum ketemu }
   while (Atas ≠ Bawah) and (not Found) do
      Tengah ← (Atas + Bawah) div 2
      depend on (T, Tengah, X)
         X = T_Tengah : Found ← true; IX ← Tengah
         X < T_Tengah : Bawah ← Tengah - 1
         X > T_Tengah : Atas ← Tengah + 1
   { Atas > Bawah  or  Found }
   { Harga Found menentukan hasil pencarian }
```

Nilai Ekstrem

1. Tuliskanlah algoritma yang menentukan nilai maksimum dan indeks tebesar pada suatu tabel jika nilai maksimum tersebut muncul lebih dari satu kali.
   Contoh: jika isi tabel adalah:

1	33	2	4	1	10	33	4	6	7
1	2	3	4	5	6	7	8	9	10

   Maka hasilnya : Maksimum = 33 pada indeks = 7.

2. Tuliskanlah algoritma yang menentukan nilai maksimum jumlah kemunculan nilai maksimum tersbut. Jika isi tabel seperti pada soal 1, maka hasilnya :
   Maksimum = 33 dan muncul 2 kali.

3. Modifikasi setiap versi algoritma maksimum untuk menghasilkan algoritma mencari nilai minimum pada tabel.

4. Tuliskanlah sebuah algoritma yang sekaligus menentukan nilai maksimum dan nilai minimum.

5. Tuliskanlah algoritma yang hasilnya adalah semua posisi di mana nilai maksimum muncul.

6. Tuliskanlah sebuah prosedur EXTREM, yang diparametrisasi sehingga prosedur yang sama dapat dipakai untuk mencari harga maksimum maupun nilai minimum. Berikan contoh pemanggilannya.

7. Karcis KA

Buatlah program untuk mengalokasikan tempat dan mencetak karcis kereta api, jika sebuah nomor kereta api dari kota X ke kota Y terdiri dari N buah gerbong kelas executive berkeapasitas NEX tempat duduk dan M gerbong kelas bisnis berkapasitas MBI tempat duduk. Tempat duduk di kereta api diatur sehingga setiap nomor terdiri dari A,B,C,D, dengan catatan bahwa A berdampingan dengan B, C dengan D. Peminta karcis maksimum boleh membeli 4 karcis dan prioritas tempat adalah harus diusahakan berdampingan, dalam satu gerbong. Formulasikan persoalan ini dengan lebih persis, dan rancanglah struktur data yang sesuai untuk persoalan ini sehingga alokasi tempat adalah optimal dan cepat. Bagaimana jika orang bisa memilih "Jendela" dan "Gang" ?

8. Parkir Mobil

Parkir mobil di suatu Plaza ingin dikomputerisasi. Tempat parkir tersebut menampung 300 mobil (maksimal) yang mempunyai identitas 1 s/d 300. Jumlah mobil yang sudah ada di dalam tempat parkir selalu dipasang pada layar besar di depan gerbang masuk. Aturan parkir adalah Rp. 300,- untuk satu jam pertama, dan Rp. 200 untuk jam berikutnya. Parkir yang kurang dari 1 jam pembayarannya dibulatkan ke atas (dihitung sebagai satu jam).

Sebuah mobil hanya boleh masuk jika dan hanya jika masih ada tempat parkir yang kosong. Tuliskanlah algoritma yang program utamanya adalah untuk mengendalikan empat buah prosedur: INISIALISASI, MOBIL-IN, MOBIL-OUT dan TERMINASI.

- Prosedur INISIALISASI adalah prosedur yang harus diaktifkan ketika tempat parkir mulai dibuka.
- Prosedur MOBIL-IN harus mampu untuk mencatat jam mulai parkir, mengalokasikan mobil pada salah satu tempat kosong dan meng-update layar besar pada gerbang masuk yang menunjukkan jumlah mobil yang sudah ada di dalam tempat parkir. Alokasi tempat kosong yang mungkin adalah: identitas paling kecil, paling dekat pintu masuk eskalator plaza, tempat sebelah "tetangganya" lebar, random, dan sebagainya. Paling masuk akal adalah bahwa alokasi ini ditentukan sendiri oleh pemakai, dan program menerima input (dari sensor yang dipasang pada setiap tempat parkir), yang harus mengupdate tabel keberadaan setiap mobil.
- Prosedur MOBIL-OUT harus mampu untuk mencetak slip pembayaran jam parkir mobil yang keluar, membebaskan tempat yang dialokasi, dan meng-update layar besar pada gerbang masuk yang menunjukkan jumlah mobil yang sudah ada.
- Prosedur TERMINASI mencetak resume pendapatan parkir hari itu dan *occupancy rate* (pemakaian tempat rata-rata: jumlah mobil parkir dibagi jumlah tempat parkir yang tersedia).

Andaikata untuk pencatatan jam disediakan sensor yang dapat mengambil current clock, tentukan di mana saja harus dipasang, supaya *key-in* petugas seminimal

mungkin, bahkan kalau perlu tidak usah *key-in* sama sekali.

Tentukan prosedur manual yang menyertai program komputer tempat parkir ini, karena program ini hanya dapat berfungsi baik jika prosedur manual nya dijalankan dengan patuh (misalnya pemberian tanda parkir, pembayaran, pencatatan jam dan bukti parkir, dan sebagainya).

Jika setiap tempat tidak diberi identitas, artinya tempat parkir dipandang sebagai satu tempat yang menampung 300 mobil tanpa peduli identitas masing-masing tempat; apa yang berubah pada program ini? Tuliskanlah kembali algoritmanya.

Sorting

1. Untuk setiap metoda sort, coba analisis secara kualitatif apa yang dikerjakan jika:
   - Elemen tabel sudah terurut
   - Elemen tabel terurut terbalik
   - Elemen tabel bernilai sama.

2. Carilah performansi keempat algoritma pengurutan yang dibahas, dan sebutkan kasus terbaik dan kasus terjeleknya.

3. Bagaimana jika pada Insertion sort diterapkan pencarian biner untuk mencari tempat penyisipan?

4. Loncat Indah

Penilaian suatu lomba loncat indah ingin ditayangkan melalui layar besar, yang setiap kali mencetak ranking, nama-nama peloncat dan score akhirnya secara terurut mulai dari yang paling tinggi s/d yang paling rendah.

Layar besar tersebut harus di-update setiap kali seorang peloncat selesai melakukan 1 kali loncatan. Seorang peloncat berhak untuk melakukan 3 kali loncatan, dan nilai yang diambil adalah nilai terbesar. Untuk mempercepat pertandingan, 3 kali loncatan tersebut tidak dilakukan berturut-turut, tetapi peloncat yang siap boleh langsung melakukan gerakan. Untuk mempercepat proses (?), pada setiap nama peloncat diberikan nomor peserta.

Tuliskanlah algoritma untuk menerima nomor peserta dan nilai loncatan, serta meng-update layar. Algoritma harus mendeteksi apakah loncatan seseorang masih dalam batasan yang diperbolehkan (maksimal 3 kali); seorang peloncat yang mencoba untuk meloncat lebih dari 3 kali harus di-diskualifikasi. daftar peserta yang terkena diskualifikasi juga harus ditayangkan (di layar lain).

Studi Representasi Tabel

Perhatikanlah pada semua prosedur dan program untuk Search dan Sort, dipakai kamus umum sebagai berikut:

```
KAMUS UMUM
   constant Nmax : integer = 100
   type TabInt : array [1..Nmax+1] of integer

{ Jika diperlukan sebuah tabel, maka akan dibuat deklarasi sebagai berikut }
   T : TabInt    { tabel integer }
   N : integer   { indeks efektif, 1 ≤ N ≤ Nmax+1 }
```

Jika prosedur diparametrisasi seperti pada spesifikasi yang diberikan, maka T dan N menjadi dua buah parameter. Padahal, nilai T dan N sebanrnya erat kaitannya satu sama lain. Deklarasi TabInt akan "lebih baik" jika "dibungkus" menjadi sebuah type komposisi sebagai berikut:

```
KAMUS UMUM
   constant Nmax : integer  = 100
   type TabInt : < TI : array [1..Nmax+1] of integer,
                   N  : integer  { indeks efektif }
                   { maksimum tabel yang terdefinisi, 1 ≤ N ≤ Nmax+1 }

{ Jika diperlukan sebuah tabel, maka akan dibuat deklarasi sebagai berikut }
    T : TabInt     { tabel integer }
```

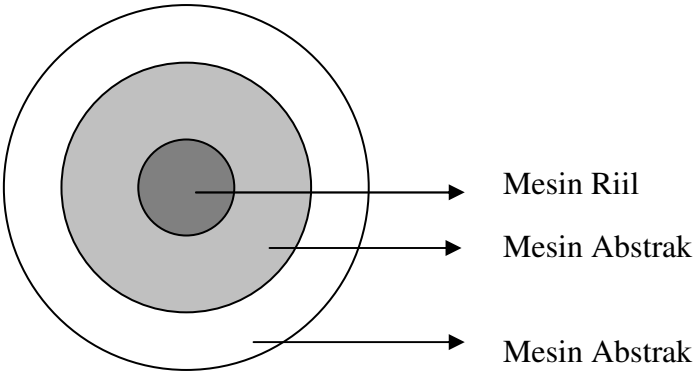
Sebagai latihan, tuliskan ulang semua prosedur yang pernah didefinisikan dengan deklarasi type komposisi ini.

MESIN ABSTRAK

Mesin adalah mekanisme yang terdefinisi dan mengerti serta mampu untuk mengeksekusi aksi-aksi primitif yang terdefinisi untuk mesin tersebut.

Mesin abstrak adalah mesin yang dianggap ada, dan diasumsikan mampu melakukan mekanisme yang didefinisikan untuk mesin tersebut. Mesin abstrak memodelkan suatu semesta (*universe*) tertentu.

Dalam pemrograman, seringkali pemrogram harus mendefinisikan mesin-mesin abstrak sebelum menuliskan kode program karena mesin abstrak memungkinkan pemrogram untuk melakukan pemecahan masalah secara bertahap. Mesin abstrak yang "diciptakan" pada tahap konseptual bahkan memungkinkan pemrogram untuk berpikir tahap demi tahap, sampai akhirnya dijabarkan dalam terminologi mesin riil.



Sebetulnya, bahasa tingkat tinggi adalah "mesin abstrak" bagi *assembler*, dan lebih abstrak bagi mesin riil yaitu komputer.

Mendefinisikan mesin abstrak, berarti mendefinisikan:
- sekumpulan state yang mungkin,
- sekumpulan **aksi primitif** yang diasumsikan dapat dimengerti dan dieksekusi mesin yang bersangkutan.

Pada diktat ini, akan diberikan contoh-contoh macam-macam mesin abstrak, yaitu:
- mesin gambar,
- mesin karakter,
- mesin integer,
- mesin rekam.

Setiap mesin abstrak tersebut akan memberikan gambaran penyelesaian persoalan dari yang sederhana sampai yang rumit melalui satu atau beberapa tahapan abstraksi.

Mesin Gambar

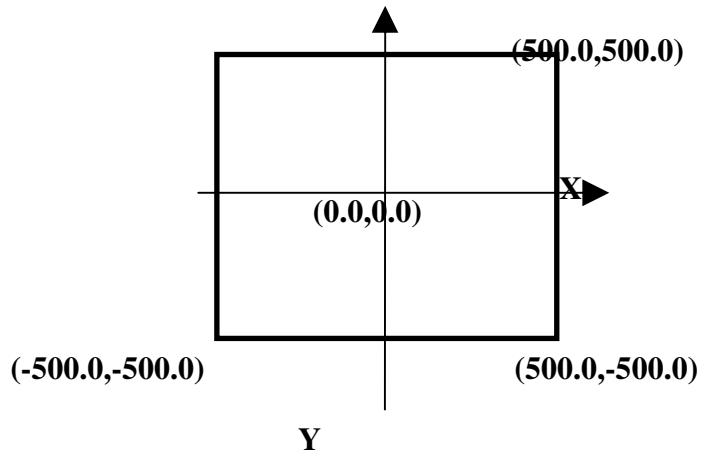
Model "State"

Definisi

Mesin gambar adalah mesin abstrak yang terdiri dari: bidang gambar dan pena. Keduanya menyatakan **keadaan** (***state***) dari mesin.

Bidang gambar:

yaitu sekumpulan titik yang membentuk sebuah permukaan terbatas yang posisinya dikenali melalui koordinat kartesian dengan (0.0,0.0) di kiri atas bidang gambar. Keadaan bidang gambar dinyatakan oleh titik-titik koordinat yang hitam atau putih Setiap titik koordinat (x,y) pada bidang gambar dapat berada dalam keadaan hitam atau putih.



Pena:

Pena mempunyai 3 atribut yaitu :

- posisi penulisan (disingkat **Pen**) : bisa "**on**" atau "**off**". Hanya pada posisi "on", pena dapat menghitamkan titik bidang gambar. Pena tidak pernah bisa memutihkan titik bidang gambar.
- posisi pada permukaan (disingkat **Posxy**): posisinya pada koordinat bidang gambar, bertype:
  type Point : < X : real**, Y :** real **>** { menyatakan koordinat real pada sumbu kartesian }
- arah pena (disingkat **Dir**): yaitu arah pena terhadap sumbu Ox seperti dalam goneometri, besarnya dinyatakan dalam derajat [-360.0..+ 360.0] derajat.

Untuk menentukan harga dari keadaan-keadaan pena ini, kita dapat memakai konstanta atau nama y (yang didefiniskan dalam kamus). Contoh:

Posxy = <10.0,20.0>, pen= off, Dir=-30.0
Posxy = <Px,Py>, pen= CurrentPos, Dir=90.0,
dengan px dan py yang telah didefinisikan dalam kamus

Mesin ini memungkinkan mekanisme yang melaksanakan beberapa aksi primitif untuk membuat gambar grafik sederhana yang didefinisikan berikut ini:

Primitif untuk mengubah posisi pena dan bidang gambar sekaligus:

```
procedure Clear
{ Aksi yang membersihkan bidang gambar, tanpa mengubah posisi pena.
  Semua titik koordinat diset menjadi putih.
  I.S. : Pen = Pospen, PosXy = <x,y>, Dir = d
  F.S. : Pen = Pospen, PosXy = <x,y>, Dir = d;
         semua titik bidang gambar menjadi putih
}
```

```
procedure Restart
{ Aksi yang membersihkan bidang gambar, dan menaruh pena pada posisi (0,0)
  dan off, Semua titik koordinat diset menjadi putih.
  I.S. : Pen = Pospen, PosXy = <x,y>, Dir = d
  F.S. : Pen = off, PosXy = <0.0,0.0>, Dir = 0.0;
         dan semua titik bidang gambar menjadi putih
}
```

Primitif untuk mengubah posisi pena terhadap bidang gambar:

```
procedure MOVE (input L : real >0.0)
{ Pena maju sejauh L satuan koordinat dengan arah yang sedang dipunyainya.
  Pena akan menghitamkan titik bidang gambar sambil maju, jika berada pada
  posisi "on".
  I.S. :  Pen = Pospen, PosXy = <x,y>, Dir = d
  F.S. :  Pen = Pospen, PosXy = <x+L cos(d), y+L sin(d)>, Dir = d
}
```

```
procedure TOPOS (input x, y : real[-500.0 .. 500.0])
{ Pena menuju posisi (x,y), tanap berubah arah dan tanpa berubah posisinya
  terhadap permukaan
  I.S. : Pen = Pospen, PosXy = <x1,y1>, Dir = d
  F.S. : Pen = Pospen, PosXy = <x,y>, Dir = d
}
```

Primitif untuk mengubah posisi pena terhadap permukaan bidang gambar:

```
procedure UP
{ Pena diposisikan pada pada posisi "off"; Pen = off
  I.S. :  Pen = Pospen, PosXy = <x,y>, Dir = d
  F.S. :  Pen = off, PosXy = <x,y>, Dir = d
}
```

```
procedure DOWN
{ Pena diposisikan pada pada posisi "on"; Pen = on
  I.S. :  Pen = Pospen, PosXy = <x,y>, Dir = d
  F.S. :  Pen = on, PosXy = <x, y>, Dir = d
}
```

Primitif untuk mengubah arah pena:

```
procedure RIGHT (input d : real[0.0..360.0])
{ Pena dibelokkan searah jarum jam sebesar d derajat.
  I.S. : Pen = Pospen, PosXy = <x,y>, Dir = d1
  F.S. : Pen = Pospen, PosXy = <x, y>, Dir = (d1-d) modreal 360.0
}
```

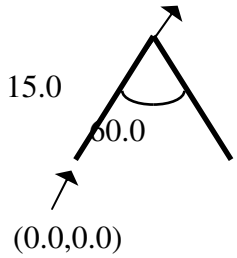
```
procedure LEFT (input d : real[0.0..360.0])
{ Pena dibelokkan berlawanan arah jarum jam sebesar d derajat.
  I.S. : Pen = Pospen, PosXy = <x,y>, Dir = d1
  F.S. : Pen = Pospen, PosXy = <x,y>, Dir = (d1+d)/360.0
}
```

```
procedure SETDIR (input d : real[0.0..360.0])
{ Pena diarahkan pada sudut sebesar d derajat.
  I.S. : Pen = Pospen, PosXy = <x,y>, Dir = d1
  F.S. : Pen = Pospen, PosXy = <x,y>, Dir = d
}
```

Studi Kasus

I. Topi

1. Buatlah algoritma yang menggambar topi sebagai berikut, mulai dari titik paling kiri yang berkoordinat (0.0,0.0)




```
Program GambarTOPI
{ I.S. : Pen = sembarang, PosXy = sembarang, Dir = sembarang }
{ F.S. : Pen = sembarang, PosXy = sembarang, Dir = sembarang }
{ Proses : sebuah gambar seperti pd Gambar di atas tergambar di layar }
```

```
KAMUS
   procedure TOPI
   { I.S. : Pen =on, PosXy = <x,y>, Dir = 60.0 }
   { F.S. : Pen= on, PosXy = <x+15.0,0.0>, Dir = -60.0
            dan topi seperti pada Gb. tergambar }
```

```
ALGORITMA
    Restart
    Down; SetDir(60.0)
    Topi
```

```
procedure TOPI
{ I.S. : Pen = on, PosXy = <x,y>, Dir = 60.0}
{ F.S. : Pen = on, PosXy = <x+ 15.0,0.0>, Dir = -60.0
         dan topi seperti pada Gb. tergambar }
```

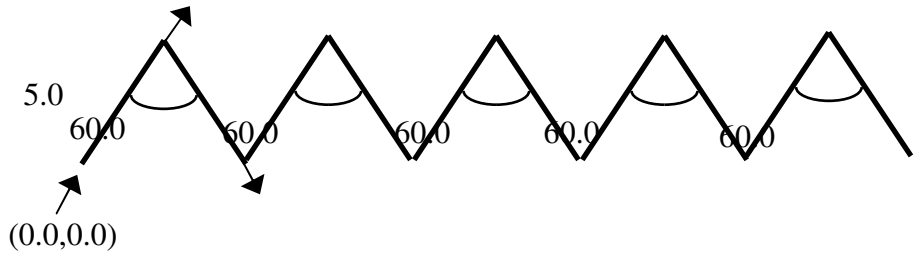
```
KAMUS LOKAL
```

```
ALGORITMA
   Move(15.0)
   Right(120.0)
   Move(15.0)
```

Gambar topi tersebut selalu berarah dan berukuran yang sama. Diinginkan sebuah prosedur TOPI yang lain, yang dapat menggambarkan topi dengan sisi yang merupakan masukan sehingga didapatkan topi dengan "ukuran" yang bermacam-macam sesuai dengan suatu nilai yang diberikan pada saat penggambaran. Maka sisi menjadi parameter dari prosedur TopiBis.

2. BarisanTopi

Buatlah algoritma yang membaca titik awal sebuah topi dan panjang sisi miringnya, kemudian menggambarkan barisan topi sebagai berikut:




```
Program BARISANTOPI
{ I.S.: Pen = sembarang, PosXy = sembarang, Dir =sembarang  }
{ F.S.: 5 buah gambar seperti pada Gambar di atas tergambar di layar }
```

```
KAMUS
   Sisi : real > 0.0     { panjang sisi sebuah topi }
   Arah : real           { arah sisi miring topi }
   type point : < X : real, Y : real >
   Awal : point          { titik awal penggambaran topi yang pertama }
   procedure TOPIBIS (input L : real)
   { I.S. : Pen = on, PosXy = <x,y>, Dir = Arah }
   { F.S. : Pen = on, PosXy = <x+ L cos(Arah), y + L sin(arah)>,
            Dir = Arah - 120.0 dan sebuah topi seperti tergambar }
```

```
ALGORITMA
   Clear
   input (Sisi, Awal,Arah)
   SetDir(Arah)
   ToPos(Awal)
   Down
   repeat 5 times
      TopiBis (Sisi)
      Left (120.0)
```

```
procedure TOPIBIS (input L : real)
{ I.S. : Pen = on, PosXy = <x,y>, Dir = Arah }
{ F.S. : Pen = on, PosXy = <x+ L cos(Arah),y + L sin(arah)>,
         Dir = Arah-120.0,  dan sebuah topi seperti pada Gb. tergambar }
```

```
KAMUS LOKAL
```

```
ALGORITMA
    Move(L)
    Right(120.0)
    Move(L)
```

Bagaimana jika "arah" topi juga ingin diparametrisasi ?

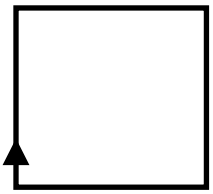
II. Bujur Sangkar

Didefinisikan type terstruktur untuk mewakili bujur sangkar:

type BujurSangkar : < P : Point; Sisi, Arah : real >
Konstanta :

< <0.0,0.0>, 5.0, 90.0 > { artinya bujur sangkar pada Gambar (1) }
< <0.0,5.0>, 15.0, 45.0 > { artinya bujur sangkar pada Gambar (2) }

1. Tuliskanlah algoritma yang membaca data bertype BujurSangkar, dan menggambarkan:



```
Program GamBarBSangkar
{ Input : Sebuah data Bujur Sangkar }
{ Proses : Menggambarkan bujur sangkar  }
{ Output : Bujur sangkar dengan awal Point bersisi Sisi berarah Arah
digambarkan  }
```

```
KAMUS
    BS : BujurSangkar
    procedure DrawSimpleRect (input L : real)
    { Menggambarkan bujur sangkar bersisi L }
    { I.S. : Pen = on, PosXy = P, Dir = Arah dari Bujur Sangkar }
    { F.S. : Pen = on, PosXy = P, Dir = Arah dari Bujur Sangkar }
```

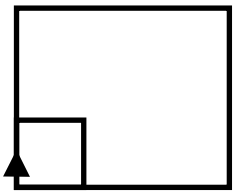
```
ALGORITMA
    input(BS)
    Restart
    SetDir(BS.Arah)
    Move(BS.Point)
    Down
    DrawSimpleRect(BS.Sisi)
```

```
procedure DrawSimpleRect (input L : real)
{ Menggambarkan bujur sangkar bersisi L }
{ I.S. : Pen = on, PosXy = P, Dir = Arah dari Bujur Sangkar }
{ F.S. : Pen = on, PosXy = P, Dir = Arah dari Bujur Sangkar }
```

```
KAMUS LOKAL
    C : BujurSangkar
```

```
ALGORITMA
    repeat 4 times
        Move(L)
        Right(90.0)
```

2. Tuliskanlah algoritma DrawTwoRect yang membaca data BujurSangkar terluar, memakai prosedur DrawSimpleRect yang didefinisikan di atas dan menggambarkan :

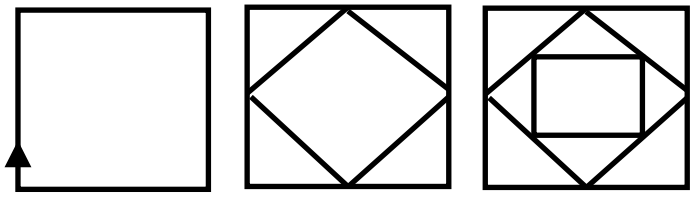


```
procedure DrawTwoRect
{ Menggambarkan 2 bujur sangkar seperti pada Gb. }
{ I.S. : Pen = on, PosXy = P, Dir = Arah dari Bujur Sangkar }
{ F.S. : Pen = on, PosXy = P, Dir = Arah dari Bujur Sangkar }
```

```
KAMUS LOKAL
```

```
ALGORITMA
    input (BS)
    Restart
    SetDir(BS.Arah); MoveTo(BS.Point)
    Down
    DrawSimpleRect(BS.Sisi); DrawSimpleRect(BS.Sisi/2)
```

3. Tuliskanlah algoritma DrawRectN yang membaca data bertype BujurSangkar, memakai prosedur DrawSimpleRect yang didefinisikan di atas dan menggambarkan :



N=1 **N=2** **N=3, dst**

```
Program DrawRectN
{ Menggambarkan bujur sangkar bersisi L, dengan level N }
{ I.S. : Pen = on, PosXy = P, Dir = Arah dari Bujur Sangkar data }
{ F.S. : Pen = on, PosXy = P, Dir = Arah dari Bujur Sangkar data }
```

```
KAMUS
    BS : BujurSangkar
    N : integer
    procedure DrawRect (input N : integer, input L : real)
    { Menggambarkan bujur sangkar bersisi L, dengan level N }
    { I.S. : Pen = on, PosXy = P, Dir = Arah dari Bujur Sangkar data }
    { F.S. : Pen = on, PosXy = P, Dir = Arah dari Bujur Sangkar data }
```

```
ALGORITMA
    input (BS,N)
    Restart
    SetDir(BS.Arah)
    MoveTo(BS.Point)
    Down
    DrawRect(N, BS.Sisi)
```

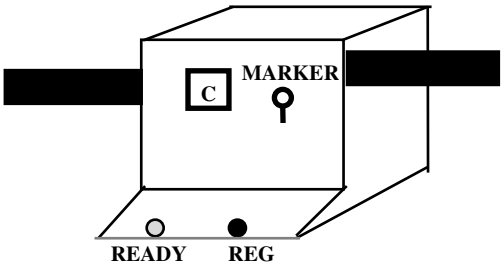
```
procedure DrawRect (input N : integer, input L : real)
{ Menggambarkan bujur sangkar bersisi L, dengan level N }
{ I.S. : Pen = on, PosXy = P, Dir = Arah dari Bujur Sangkar data }
{ F.S. : Pen = on, PosXy = P, Dir = Arah dari Bujur Sangkar data }
```

```
KAMUS LOKAL
```

```
ALGORITMA
    Move(L/2); Right(45)
    if (N ≠ 1) then
        DrawRect(N-1, (L/2) * √2)
    Left (45.0); Move(L/2); Right(90.0)
    repeat 3 times
        Move(L); Right (90.0)
```

Mesin Rekam

Mesin REKAM adalah mesin abstrak yang terdiri dari : pita dan tombol READY, REG dan sebuah tombol MARKER.
Mesin rekam adalah mesin yang dipakai untuk merekam pita karakter yang akan dibaca oleh mesin karakter.



Primitif untuk mengubah posisi pita:

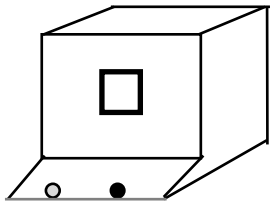
```
procedure READY
{ Mesin disiapkan, pita disiapkan untuk direkam.
  Jendela siap ditulisi sebuah karakter yang akan direkam pada posisi
  pertama.
  I.S. : sembarang
  F.S. : posisi perekaman pertama
}
```

```
procedure REG
{ Merekam dan posisi pita yang siap direkami dimajukan satu karakter.
  I.S. : Karakter pada jendela = CC, CC ≠ '.'
  F.S. : CC direkam pada posisi perekaman, pita maju satu karakter.
}
```

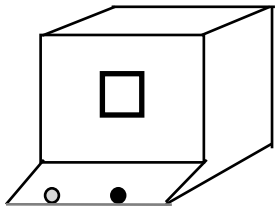
```
procedure MARKER
{ Pita direkami dengan tanda akhir pita (mark).
  I.S. : Karakter pada jendela = CC, CC ≠ '.'
  F.S. : CC ≠ '.', tombol MARKER diaktifkan. Mesin dimatikan.
}
```

Mesin Integer (Pencacah)

Mesin integer adalah mesin abstrak yang terdiri dari: tombol **RESET** dan **INC**. Seperti halnya mesin karakter, mesin ini sebuah jendela yang menunjukkan sebuah angka integer yang diingatnya, yang disebut CI (Current Integer). Mesin ini berfungsi sebagai pencacah bilangan integer.



Setelah tombol RESET ditekan



Setelah 5 kali menekan tombol INC

Primitif untuk mengubah keadaan mesin:

```
procedure RESET
{ Pencacah disiapkan untuk dipakai, Harga bilangan bulat yang
  disimpan adalah Nol.
  I.S. : sembarang
  F.S. : CI = 0
}
```

```
procedure INC
{ Pencacah dimajukan satu
  I.S. : CI = Harga
  F.S. : CI = Harga + 1
}
```

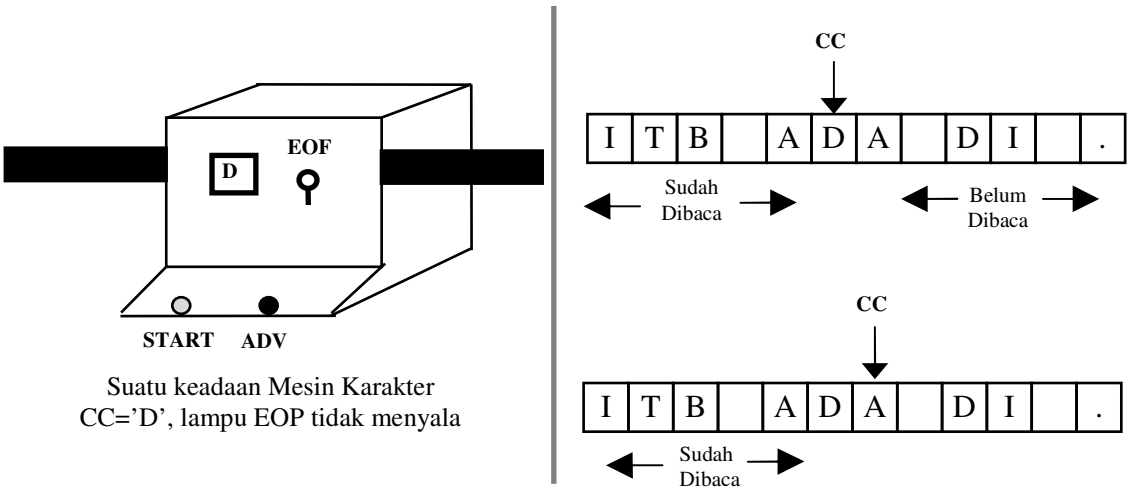
Mesin Karakter

(Model akses sekuensial)

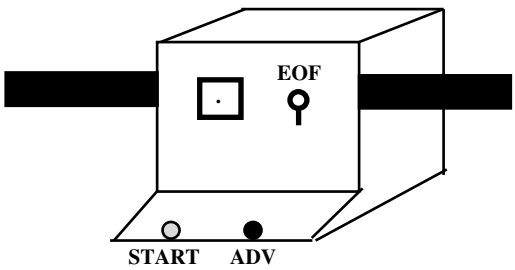
Definisi

Mesin karakter adalah mesin abstrak yang terdiri dari :
- pita berisi deret karakter, yang diakhiri dengan '.' (titik); pita yang hanya berisi '.' disebut sebagai pita kosong,
- tombol **START**, **ADV**,
- sebuah Lampu **EOP** (End Of Pita),
- "jendela" yang ukurannya sebesar satu karakter, hanya karakter yang posisinya sedang pada jendela dapat dikonsultasi (dibaca); karakter lain tidak kelihatan. Karakter yang sedang pada jendela dinamakan **CC** (Current Character).

Mesin mempunyai mekanisme untuk mengubah posisi pita dan menyalakan lampu EOP jika karakter yang ada pada jendela adalah titik. Keadaan (*state*) dari mesin setiap saat ditentukan oleh CC dan lampu **EOP**. Tombol **START** dan **ADV** digunakan untuk mengubah *state* mesin. Mesin hanya dapat dioperasikan jika **EOP** tidak menyala.



Suatu keadaan Mesin Karakter
CC='D', lampu EOP tidak menyala



Ketika CC='.', lampu EOP menyala

Primitif untuk mengubah posisi pita:

```
procedure START
{ Mesin siap dioperasikan. Pita disiapkan untuk dibaca.
  Karakter pertama yang ada pada pita posisinya adalah pada jendela
  I.S. : sembarang
  F.S. : CC adalah karakter pertama pada pita
         Jika CC ≠ '.' maka EOP akan padam (false)
         Jika CC = '.' maka EOP akan menyala (true)
}
```

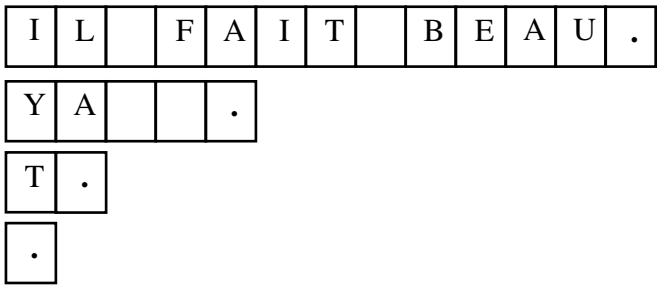
```
procedure ADV
{ Pita dimajukan satu karakter.
  I.S. : Karakter pada jendela = CC, CC ≠ '.'
  F.S. : CC adalah karakter berikutnya dari CC yang lama,
         CC mungkin = '.'
         Jika  CC = '.' maka EOP akan menyala (true)
}
```

Catatan:

EOP diwakili oleh boolean, bernilai true jika menyala; atau false jika tidak menyala. Jika **EOP** menyala, mesin sudah tidak dapat dioperasikan lagi.

Studi Kasus Mesin Karakter

1. Diberikan empat pita sbb., HITUNG-KARAKTER dari pita-pita sbb.:



Deretan aksi (penekanan tombol) pada kedua mesin untuk menghitung banyaknya huruf pada pita:

1a) **START**; **RESET** { CC='I', CI=0 }
ADV ; INC { CC='L', CI=1 }
ADV ; INC { CC=' ', CI=2 }
ADV ; INC { CC='F', CI=3 }
ADV ; INC { CC='A', CI=4 }
ADV ; INC { CC='I', CI=5 }
ADV ; INC { CC='T', CI=6 }
ADV ; INC { CC=' ', CI=7 }
ADV ; INC { CC='B', CI=8 }
ADV ; INC { CC='E', CI=9 }
ADV ; INC { CC='A', CI=10 }
ADV ; INC { CC='U', CI=11 }
ADV ; INC { CC='.', CI=12 }

1b) **START** ; **RESET**
INC ; ADV
INC ; ADV
INC ; ADV
INC ; ADV
INC ; ADV
INC ; ADV
INC ; ADV
INC ; ADV
INC ; ADV
INC ; ADV
INC ; ADV
INC ; ADV

2) **START** ; **RESET** { CC='Y', CI=0 }
ADV ; INC { CC='A', CI=1 }
ADV ; INC { CC=' ', CI=2 }
ADV ; INC { CC=' ', CI=3 }
ADV ; INC { CC='.', CI=4 }

3) **START** ; **RESET** { CC='T', CI=0 }
ADV ; INC { CC='.', CI=1 }

4) **START** ; **RESET** { CC='.', CI=0 }

Diskusi:

- Apa beda antara solusi 1a) dan 1b)?
- Bagaimana mendeduksi iterasi dari deretan aksi di atas?
- Kesulitan penggunaan repeat n times pada ksus ini dibandingkan dengan kasus algoritma BUJURSANGKAR, di mana iterasi dideduksi dengan mudah menjadi repeat 4 times. Sebenarnya repeat N times bisa dipakai pada kasus ini, yaitu dengan menghitung dulu banyaknya huruf, baru repeat sebanyak banyaknya huruf. Tapi bodoh sekali!
- Pengenalan notasi baru: skema iterasi (ingat: bukan *loop* sederhana seperti dalam bahasa pemrograman). Cukup diperkenalkan model dengan mark saja.

2. COUNTHURUF

Diberikan sebuah mesin karakter dengan pita berisi karakter (mungkin kosong). Buatlah algoritma untuk menghitung banyaknya huruf yang ada pada pita tersebut dengan mesin integer. **Banyaknya** karakter pada pita kosong adalah nol.

```
Program COUNTHURUF
{ SKEMA PEMROSESAN DENGAN MARK : menghitung banyaknya huruf pada pita
karakter, memakai mesin pencacah }

KAMUS

ALGORITMA
   RESET                 { Inisialisasi, CI = 0 }
   START                 { First_Elmt }
   while (CC ≠ '.') do   { not EOP}
      INCR               { Proses : CI = CI + 1 }
      ADV                { Next_Elmt }
   { CC = '.' }
   output (CI)           { Terminasi}
```

3. HITUNG-A

Diberikan sebuah mesin karakter dengan pita berisi karakter (mungkin kosong), Buatlah algoritma untuk menghitung banyaknya huruf 'A' yang ada pada pita tersebut dengan mesin integer.

```
Program COUNT_A
{ SKEMA PEMROSESAN DENGAN MARK : menghitung banyaknya huruf A pada pita
karakter, memakai mesin pencacah }

KAMUS

ALGORITMA
   RESET                 { Inisialisasi, CI = 0 }
   START                 { First_Elmt }
   while (CC ≠ '.') do   { not EOP }
      depend on CC       { Proses }
         CC = 'A' : INCR
         CC ≠ 'A' : -
      ADV                { Next_Elmt }
   { CC = '.' }
   output (CI)           { Terminasi }
```

Penjelasan :

Banyaknya karakter ‘A’ pada pita kosong adalah nol.

```
Program COUNT_A-2
{ SKEMA PEMROSESAN dengan penanganan kasus kosong }
{ memakai mesin pencacah }

KAMUS

ALGORITMA
   START           { First_Elmt }
   depend on CC
      CC = '.' :  output ("pita kosong")
      CC ≠ '.' :
                  RESET     { Inisialisasi, CI = 0 }
                  repeat
                     if (CC = 'A') then
                        INCR
                     else   { CC ≠ 'A' → aksi kosong }
                        -
                     ADV    { Next_Elmt }
                  until (CC = '.')
                  output (CI) {Terminasi}
```

Penjelasan :
Banyaknya karakter ‘A’ pada pita kosong tidak terdefinisi, karena itu pita kosong ditangani secara khusus.

4. FREKUENSI HURUF ‘A’

Diberikan sebuah mesin karakter dengan pita berisi karakter (mungkin kosong), Buatlah algoritma untuk menghitung frekuensi relatif huruf 'A' yang ada pada pita tsb., Frekuensi relatif huruf 'A' adalah banyaknya huruf 'A' dibandingkan banyaknya seluruh karakter yang ada pada pita karakter.

```
Program FREKA-1
{ SKEMA PEMROSESAN dengan Mark, tanpa penanganan kasus kosong }
{ tidak memakai mesin pencacah, melainkan menyimpan dalam nama }
{ yang didefinisikan pada Kamus }
```

```
KAMUS
   CPT_KAR : integer     { banyaknya karakter yang sudah dibaca }
   CPTA : integer        ( banyaknya huruf A yang muncul
                           pada bagian pita yang sudah dibaca }
```

```
ALGORITMA
   CPT_KAR ← 0     { Inisialisasi  }
   CPTA ← 0        { Inisialisasi }
   START           { First_Elmt }
   while (CC ≠ '.') do
      CPT_KAR ← CPT_KAR + 1
      if (CC = 'A') then
         CPTA ← CPTA + 1
      else   { CC ≠ 'A' → aksi kosong }
         -
      ADV          { Next_Elmt }
   { CC = '.' : semua karakter pada pita sudah dibaca, mungkin CPT_KAR = 0 }
   { Terminasi }
   if (CPT_KAR ≠ 0) then
      output (CPTA/CPT_KAR)
   else  { CPT_KAR = 0 }
      output ('Frekuensi tidak terdefinisi')
```

Penjelasan :

1. Banyaknya karakter ‘A’ pada pita kosong tidak terdefinisi.
2. Banyaknya karakter pada pita kosong adalah nol.
3. Perhatikan invarian CC, CPTA dan CPT_KAR yang selalu benar.
4. Pembagian dengan nol dihindarkan dengan analisis kasus pita kosong atau tidak pada bagian TERMINASI.
5. Program benar, tapi pemilihan skema kurang tepat.

```
Program FREKA-2
{ SKEMA PEMROSESAN dengan Mark, dengan penanganan kasus kosong }
{ tidak memakai mesin pencacah }

KAMUS
   CPT_KAR : integer     { banyaknya karakter pd pita yang sudah dibaca }
   CPTA : integer        { banyaknya huruf A yang muncul
                           pada bagian pita yang sudah dibaca }

ALGORITMA
   START                          { First_Elmt }
   if (CC = '.') then
      output ("Pita kososng")
   else  { CC ≠ '.' }
      CPT_KAR ← 0                 { Inisialisasi  }
      CPTA ← 0                    { Inisialisasi }
      repeat
         CPT_KAR ← CPT_KAR + 1
         if (CC = 'A') then
              CPTA ← CPTA + 1
         { else : CC ≠'.' ➔ do nothing }
         ADV                      { Next_Elmt }
      until (CC ='.')
      output (CPTA/CPT_KAR)       { Terminasi. CPT_KAR pasti tidak nol. }
```

Penjelasan :

1. Banyaknya karakter ‘A’ pada pita kosong tidak terdefinisi.
2. Banyaknya karakter pada pita kosong adalah nol.
3. Jika kasus kosong tidak ditangani secara khusus, maka akan terjadi pembagian dengan nol.
4. Pemilihan skema lebih baik dari pada versi pertama.

5. HITUNG-LE

Diberikan sebuah mesin karakter dengan pita berisi karakter (mungkin kosong), Buatlah algoritma untuk menghitung banyaknya pasangan dua huruf ‘LE’ yang ada pada pita tersebut.

Versi 1:

Realisasi dengan membuat mesin "*couple*" yaitu mesin yang mampu untuk menampilkan dua karakter sekaligus berdasarkan mesin karakter. Couple adalah dua buah karakter berturutan (suksesif) yang meuncul pada pita.

```
Program COUNTLE-1
{ SKEMA PEMROSESAN DENGAN MARK }
{ Solusi 1 : Mesin COUPLE }

KAMUS
   CPTLE : integer    { banyaknya 'LE' pada bagian pita yang sudah dibaca }
   C1,C2 : character  { C1, C2 adalah Couple }
   procedure START-COUPLE   { mendapatkan couple yang pertama }
   { I.S. : sembarang }
   { F.S. : Couple pertama terbentuk: C1 = ' ', C2 = CC , CC mungkin ='.' }
   procedure ADV-COUPLE     { next-couple }
   { I.S. : C1  dan C2, C2 ≠ '.'}
   { F.S. : C1 =  C2, C2 = CC, CC mungkin = '.' }

ALGORITMA
   START-COUPLE                { First_Elmt }
   CPTLE ← 0
   while (CC ≠ '.') do         { not End-Couple }
      if (C1 = 'L') and (C2 = 'E') then
         CPTLE ← CPTLE + 1     { couple "LE" }
      { else : C1 ≠ 'L' or C2 ≠ 'E' : - }
      ADV-COUPLE               { Next_Elmt }
   { CC = '.' }
   output (CPTLE)              { Terminasi }
```

```
procedure START-COUPLE
{ SKEMA PEMROSESAN DENGAN MARK, Solusi 1 : Mesin COUPLE }
{ I.S. : sembarang }
{ F.S. : Couple pertama terbentuk : C1 = ' ', C2 = CC, CC mungkin = '.' }

KAMUS LOKAL

ALGORITMA
   C1 ← ' '     { karena yang dicari adalah 'LE'.
                  Bagaimana jika yang dicari pasangan lain? }
   START
   C2 ← CC
```

```
procedure ADV-COUPLE
{ SKEMA PEMROSESAN DENGAN MARK, Solusi 1: Mesin COUPLE }
{ I.S. : C1 dan C2, C2 ≠ '.' }
{ F.S. : C1 = C2, C2 = CC, CC mungkin = '.' }

KAMUS LOKAL

ALGORITMA
   C1 ← C2
   ADV
   C2 ← CC
```

Penjelasan :

1. Solusi ini memakai Hitung A sebagai pola solusi. Pada solusi ini, couple 'LE' analog dengan sebuah huruf 'A'.
2. Banyaknya 'LE' pada pita kosong adalah nol.
3. Banyaknya 'LE' pada pita dengan satu karakter adalah nol..
4. Banyaknya 'LE' pada pita yang tidak mengandung dua huruf suksesif 'LE' adalah nol. Dalam hal ini tidak dapat dibedakan antara nilai nol pada pita kosong, pita satu karakter atau untuk pita yang tidak mengandung 'LE'.
5. Perhatikan pendefinisian couple yang pertama: C1= *Blank* dan C2 = huruf pertama pada pita. Couple pertama ini merupakan couple yang dibuat berdasarkan definisi pemrogram, bukan merupakan couple yang sesungguhnya muncul pada pita.
6. Buatlah solusi dengan mengambil couple pertama yang muncul pada pita adalah benar-benar dua karakter pertama pada pita. Solusi ini mengharuskan akses terhadap karakter pertama pita. Jika pita kosong atau mengandung hanya satu karakter, maka pita 'couple' akan merupakan pita kosong (couple pertama tidak terdefinisi).
7. Solusi dengan pendefinisian couple pertama bukan dari pita bukan solusi yang ideal, namun 'praktis'. Harap hati-hati dengan pemberian harga yang bukan berasal dari pita yang diberikan.

Versi 2 :

Idenya adalah mengingat-ingat jika menemukan 'L', dan memeriksa karakter yang selanjutnya. Di sini tidak dibentuk mesin abstrak lain seperti pada solusi 1.

```
Program COUNTLE2
{ SKEMA PEMROSESAN DENGAN MARK }
{ Solusi 2: Memorisasi satu karakter sebelum karakter yang ada di jendela }

KAMUS
   CPTLE : integer       { banyaknya 'LE' pada bagian pita yang sudah
                           Dibaca }
   Prec-Is-L : boolean   { true jika karakter sebelum CC adalah 'L' }

ALGORITMA
   Prec-Is-L ← false          { Inisialisasi }
   CPTLE ← 0                  { Inisialisasi }
   START                      { First_Elmt }
   while (CC ≠ '.') do
      Prec-Is-L ← (CC = 'L')
      ADV                     { Next_Elmt }
      if (CC = 'E') and Prec-Is-L then
         CPTLE ← CPTLE + 1
   output (CPTLE)             { Terminasi }
```

Penjelasan :

1. Solusi ini bukan mengambil pola Hitung ‘A’.
2. Banyaknya ‘LE’ pada pita kosong adalah nol.
3. Banyaknya ‘LE’ pada pita dengan satu karakter adalah nol.
4. Banyaknya ‘LE’ pada pita yang tidak mengandung dua huruf suksesif ‘LE’ adalah nol. Dalam hal ini tidak dapat dibedakan antara nilai nol pada pita kosong, pita satu karakter atau untuk pita yang tidak mengandung ‘LE’.
5. Prec-Is-L adalah sebuah boolean, dapat pula direpresentasi dengan sebuah nama bertype character, yang setiap saat berisi nilai huruf yang muncul sebelum karakter pada jendela. Inisialisasi untuk CC = karakter pertama pada pita menjadi persoalan yang sama dengan versi-1.

Versi- 3 :
Idenya adalah maju terus sampai menemukan 'L', dan memeriksa karakter yang berikutnya.

```
Program COUNTLE-3
{ SKEMA PEMROSESAN DENGAN MARK }
{ Solusi 3: majukan pita sampai ketemu 'L', periksa karakter berikutnya.
Proses ini diulang sampai seluruh karakter selesai diproses }

KAMUS
   CPTLE : integer    { banyaknya 'LE' pada bagian pita yg sudah dibaca }

ALGORITMA
   CPTLE ← 0       { Inisialisasi }
   START           { First_Elmt }
   while (CC ≠ '.') do
      { Cari sampai ketemu 'L', namun mungkin ketemu '.' }
      while (CC ≠ 'L') and (CC ≠ '.') do
         ADV
      { CC = 'L' or CC = '.' }
      if (CC = 'L') then
         ADV       { Boleh ADV, karena CC bukan '.' }
      if (CC = 'E') then CPTLE ← CPTLE + 1
      { else : CC ≠ 'L' :  - }
   { CC = '.', seluruh karakter pada pita sudah dibaca }
   output (CPTLE)  { Terminasi }
```

Penjelasan :

1. Solusi ini berdasarkan 'pencarian' karakter 'L' pada pita, dan memeriksa karakter sesudahnya. Jika karakter pada jendela = 'L', pasti pita belum berada pada mark; karena itu boleh ADV.
2. Banyaknya 'LE' pada pita kosong adalah nol.
3. Banyaknya 'LE' pada pita dengan satu karakter adalah nol.
4. Banyaknya 'LE' pada pita yang tidak mengandung dua huruf suksesif 'LE' adalah nol. Dalam hal ini tidak dapat dibedakan antara nilai nol pada pita kosong, pita satu karakter atau untuk pita yang tidak mengandung 'LE'.
5. Solusi ini bukan merupakan skema sekuensial dengan mark yang murni. Perhatikan modifikasi yang dilakukan, dan test setiap kali hendak dilakukan ADV.

Ekspresi Aritmatika

Diberikan sebuah mesin karakter dengan pita berisi karakter yang hanya terdiri dari karakter angka '0'..'9', *blank* dan diakhiri titik. Teks pada pita dapat diinterpretasikan sebagai sebuah ekspresi aritmatika dalam bentuk notasi *infix*: operan1 operator operan2. Antara operan1 dan operator atau antara operator dan operan2 boleh dipisahkan oleh 1 *blank* atau lebih dari satu *blank*. Demikian pula, sebelum operan1 dan sesudah operan2 boleh ada satu atau beberapa *blank*. Operan1 dan operan2 terdiri dari sederetan angka yang dijamin masih dalam batas perhitungan mesin pengeksekusi algoritma. Operan2 tidak mungkin nol Operator adalah salah satu dari empat operator aritmatika sebagai berikut : '*', ':', '+', '-' yang artinya pengali, pembagi, penjumlah, pengurang dua bilangan integer menjadi bilangan integer. Dijamin bahwa ekspresi pada pita tidak mengandung kesalahan sintaks dan pita tidak mungkin kosong karena pasti mengandung sebuah ekspresi.

```
Program EKSPRESIARITMATIKA
{ Spesifikasi : lihat persoalan }
```

```
KAMUS
    constant Blank : character = ' '     { karakter "blank" }
    constant Mark : character = '.'      { karakter "titik" }
    CC : integer
    Operan1, Operan2 : integer
    Operator : character
    Hasil : integer
    procedure Getoperand (output Operan : integer)    { mengambil operan }
    { I.S. : CC adalah karakter pertama operan }
    { F.S. : CC bukan angka,
             CC adalah Blank atau salah satu karakter berikut :
             '.' , '*', '+', '-', ':'
             dan operan berisi hasil harga operan integer }
    procedure Ignore_Blank   { mengabaikan satu atau beberapa Blank }
    { I.S. : CC sembarang }
    { F.S. : CC bukan blank,
             CC adalah karakter angka atau salah satu karakter sbb. :
             '.' , '*', '+', '-', ':' }
```

```
ALGORITMA
    START         { First_Elmt }
    { pita tidak mungkin kosong }
    Ignore_Blank
    { CC = karakter pertama operan1}
    GetOperan(Operan1)
    { CC = karakter Blank sesudah operan1}
    Ignore_Blank
    { CC = operator }
    Operator ← CC
    ADV
    Ignore_Blank
    { CC = karakter pertama Operan2 }
    GetOperan(Operan2)
    { CC = Mark atau Blank, tidak peduli, sisa pita tidak diproses }
    { Hitung ekspresi }
    depend on Operator
        Operator = '*' :  Hasil ← Operan1 * Operan2
        Operator = '+' :  Hasil ← Operan1 + Operan2
        Operator = '-' :  Hasil ← Operan1 - Operan2
        Operator = ':' :  Hasil ← Operan1 div Operan2
    output (Hasil)     { Terminasi }
```

```
procedure Getoperand (output Operan : integer)        { mengambil operan }
{ I.S. : CC adalah karakter pertama operan }
{ F.S. : CC bukan angka, CC adalah Blank atau salah satu karakter berikut :
         '.' , '*', '+', '-', ':' }
{ dan operan berisi hasil harga operan integer }
```

```
KAMUS LOKAL
    function KarakterToInteger (CC : character ['0'..'9']) → integer [0..9]
    { Konversi karakter angka ke integer.
      Lihat pendefinisannya pada bagian FUNGSI }
```

```
ALGORITMA
    { I.S. : CC adalah karakter pertama operan, selalu karakter angka }
    { Berikut ini adalah pemakaian skema pengulangan yang "benar" }
    { tapi kurang tepat, karena dirancang untuk dipakai kelak, jika }
    { program menangani kesalahan sintaks }
    { skema apa yang lebih baik untuk akuisisi operan dalam kasus ini? }
    Operan ← 0
    while (CC ∈ ['0'..'9']) do
        Operan ← Operan * 10 + KarakterToInteger(CC)
        ADV
    { CC ≠ ['0'..'9'] : CC ∈ ['.', '*', '+', '-', ':', blank] }
```

```
procedure Ignore_Blank   { mengabaikan satu atau beberapa blank }
{ I.S. : CC adalah blank }
{ F.S. : CC ≠ Blank, adalah karakter angka atau salah satu karakter sbb :
         '.' , '*', '+', '-', ':' }
```

```
KAMUS LOKAL
```

```
ALGORITMA
    { I.S. : CC sembarang }
    while (CC = Blank) and (CC ≠ Mark) do
        ADV
    { F.S. : CC ≠ Blank : CC ∈ ['0'..'9','.', '*', '+', '-', ':'] }
```

1. Skema "while" yang dipakai pada prosedur GetOperand kurang tepat. Pelajarilah, dan usulkan bentuk pengulangan yang lebih baik.
2. Bagaimana jika ekspresi mengandung kesalahan? Deskripsikan kesalahan yang mungkin terjadi dan penanganannya. Perhatikan bahwa algoritma akan menjadi jauh lebih rumit.

Panjang Kata Rata-Rata pada Pita

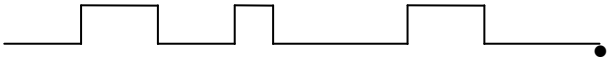
Diberikan sebuah mesin karakter dengan pita berisi karakter (mungkin kosong),yang diakhiri titik, hitunglah panjang rata-rata kata yang ada pada pita tsb. Panjang kata rata-rata tidak terdefinisi jika pita kosong atau pita tidak mengandung kata (hanya berisi '*blank*' dan titik). Kata adalah sederetan karakter suksesif pada pita yang merupakan karakter bukan *Blank*.

Ini adalah contoh soal yang memberikan gambaran tentang Sebuah pita dapat digambarkan secara skematis sebagai berikut :

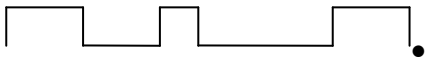
a. Hanya mengandung titik (pita kosong)

b. Hanya mengandung blank diakhiri titik

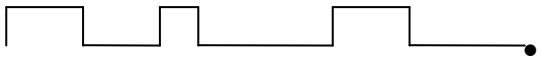
c. Mengandung blank di awal dan akhir pita



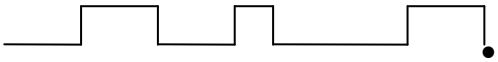
d. Tidak mengandung blank di awal maupun di akhir pita



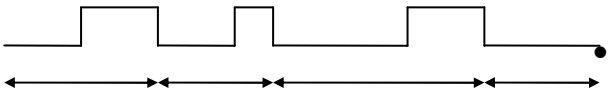
e. Mengandung blank di akhir pita



f. Mengandung blank di awal pita



Versi 1 :
Akhir dari proses adalah sebuah **boolean**, yang akan berisi true jika kata terakhir telah diakuisisi dan diproses. Kata diakuisisi mulai dari karakter pertama sesudah akhir kata (atau karakter pertama pita untuk kata pertama). Akuisisi kata terakhir menghasilkan 'kata kosong'.



```
Program PANJANGRATAKATA1
{ Versi 1 : SKEMA PEMROSESAN tanpa penanganan kasus kosong }

KAMUS
    constant Mark : character = '.'
    constant Blank : character = ' '
    LKata : integer    { panjang kata terakhir yang susah diakuisis }
    NbKata : integer   { banyaknya kata pada pita }
    LTotal : integer   { akumulasi panjang kata }
    EndKata : boolean  { true jika kata terakhir sudah diakuisisi }

    procedure Ignore_Blank   { mengabaikan satu atau beberapa blank }
    { I.S. : CC adalah sembarang }
    { F.S. : CC ≠ Blank, atau CC = Mark }

    procedure HITUNGPANJANG  { menghitung panjang kata }
    { I.S. : CC adalah karakter pertama dari kata }
    { F.S. : CC = Blank, atau CC = Mark; CC adalah karakter sesudah huruf
             terakhir kata yang diakuisisi
             LKata berisi panjang kata yang sudah diakuisisi }

    procedure STARTKATA      { mengabaikan satu atau beberapa blank }
    { I.S. : CC sembarang }
    { F.S. : EndKata = true, dan CC = Mark; }
    {        atau EndKata = false, LKata adalah panjang kata yang sudah
             diakuisisi, CC karakter pertama sesudah karakter terakhir kata }

    procedure ADVKATA
    { I.S. : EndKata = false; CC adalah karakter sesudah karakter terakhir
             dari kata yg sudah diakuisisi }
    { F.S. : EndKata = true, dan CC = Mark
             atau EndKata=false, LKata adalah panjang kata yang sudah
             diakuisisi, CC karakter pertama sesudah karakter terakhir kata }

ALGORITMA
    LTotal ← 0
    NbKata ← 0
    STARTKATA
    while not EndKata do
        LTotal ← LTotal + LKata
        NbKata ← NbKata + 1
        ADVKata
    { EndKata ➔ semua karakter sudah diakuisisi }
    if (NbKata ≠ 0) then
        output (LTotal/NbKata)
    else { NbKata = 0 }
        output ('Pita tidak mengandung kata')
```

```
procedure Ignore_Blank   { mengabaikan satu atau beberapa blank }
{ I.S. : CC sembarang }
{ F.S. : CC ≠ Blank, atau CC = Mark}

KAMUS LOKAL

ALGORITMA
    { I.S. : CC sembarang }
    while (CC = Blank) and (CC ≠ Mark) do
        ADV
    { F.S. : CC ≠ Blank or CC = Mark }
```

```
procedure HITUNGPANJANG
{ menghitung panjang kata }
{ I.S. : CC adalah karakter pertama dari kata,CC ≠ Blank dan CC ≠ Mark }
{ F.S. : CC = Blank, atau  CC = Mark; CC adalah karakter sesudah huruf
terakhir kata yang diakuisisi; LKata berisi panjang kata yang sudah
diakuisisi. }
```

KAMUS LOKAL

ALGORITMA

```
    LKata ← 1    { karena berada pada karakter pertama pita }
    iterate
        ADV
    stop (CC = Mark) or (CC = Blank)
        LKata ← LKata + 1
    { CC = Mark or CC = Blank : LKata = banyaknya karakter kata yang
      diakuisisi }
```

```
procedure STARTKATA   { mengabaikan satu atau beberapa Blank }
{ I.S. : CC sembarang }
{ F.S. : EndKata true, dan CC = Mark;
         atau EndKata=false, LKata adalah panjang kata yang sudah diakuisisi,
         CC karakter pertama sesudah karakter terakhir Kata }
```

KAMUS LOKAL

ALGORITMA

```
    START
    Ignore_Blank
    depend on CC
        CC = Mark : EndKata ← true
        CC ≠ Mark : EndKata ← false
                    HITUNGPANJANG
```

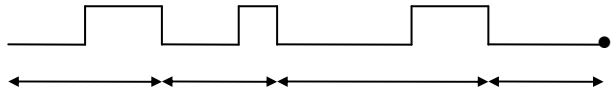
```
procedure ADVKATA  { mengabaikan satu atau beberapa Blank }
{ I.S. : EndKata = false; CC adalah karakter sesudah karakter terakhir kata
         yang sudah diakuisisi }
{ F.S. : EndKata true, dan CC = Mark
         atau EndKata=false, LKata adalah panjang kata yang sudah diakuisisi,
         CC karakter pertama, sesudah karakter terakhir kata }
```

KAMUS LOKAL

ALGORITMA

```
    Ignore_Blank
    depend on CC
        CC = Mark : EndKata ← true
        CC ≠ Mark : HITUNGPANJANG
```

Versi 2:
Akhir dari proses adalah sebuah kata yang 'kosong', yaitu panjangnya NOL. Model akuisisi kata sama dengan Versi 1.



```
Program PANJANGRATAKATA2

KAMUS
   constant  Mark  : character = '.'
   constant  Blank : character = ' '
   LKata : integer   { panjang kata terakhir yang sudah diakuisisi }
   NbKata : integer  { banyaknya kata pada pita }
   LTotal : integer  { akumulasi panjang kata }

   procedure  Ignore_Blank  { mengabaikan satu atau beberapa Blank }
   { I.S. : CC adalah sembarang }
   { F.S. : CC ≠ Blank, atau CC = Mark }

   procedure HITUNGPANJANG  { menghitung panjang kata }
   { I.S. : CC adalah karakter pertama dari kata }
   { F.S. : CC = Blank, atau CC = Mark; CC adalah karakter sesudah huruf
   terakhir kata yang diakuisisi;  LKata berisi panjang kata yang sudah
   diakuisisi }

   procedure STARTKATA      { mengabaikan satu atau beberapa Blank }
   { I.S. : CC sembarang }
   { F.S. : LKata = 0, dan CC = Mark; }
            atau LKata ≠ 0, LKata adalah panjang kata yang sudah diakuisisi,
            CC = karakter pertama sesudah karakter terakhir kata }

   procedure  ADVKATA
   { I.S. : LKata=0, CC adalah karakter sesudah karakter terakhir kata
            yang sudah diakuisisi }
   { F.S. : LKata = 0, dan CC = Mark;
            atau LKata ≠ 0, LKata adalah panjang kata yang sudah diakuisisi,
            CC karakter pertama sesudah karakter terakhir kata }

ALGORITMA
   NbKata ← 0        { belum ada kata diakuisisi }
   LTotal ← 0        { belum ada kata diakuisisi: #kata = 0 }
   STARTKATA
   while (LKata ≠ 0) do
      LTotal ← LTotal + LKata
      NbKata ← NbKata + 1
      ADVKata
   { LKata = 0 ➔ mark }
   depend on NbKata
      NbKata ≠ 0 : output (LTotal/NbKata)
      NbKata = 0 : output ("Pita tidak mengandung kata")
```

```
procedure Ignore_Blank   { mengabaikan satu atau beberapa Blank }
{ I.S. : CC adalah sembarang }
{ F.S. : CC ≠ Blank, atau CC = Mark }
```

KAMUS LOKAL

ALGORITMA

```
    { I.S. : CC sembarang }
    while (CC = Blank) and (CC ≠ Mark) do
        ADV
    { F.S. : CC ≠ Blank or CC = Mark }
```

```
procedure HITUNGPANJANG  { menghitung panjang kata }
{ I.S. : CC adalah karakter pertama dari kata atau CC = Mark}
{ F.S. : CC = Blank, atau CC = Mark;  CC adalah karakter sesudah huruf
terakhir kata yang diakuisisi; LKata berisi panjang kata yang sudah
diakuisisi }
{ Catatan : modul ini berbeda dengan Versi 1 }
```

KAMUS LOKAL

ALGORITMA

```
    LKata ← 0
    while (CC ≠ Mark) and (CC ≠ Blank) do       { kenapa dipakai WHILE?}
        LKata ← LKata + 1
        ADV
    { CC = Mark or CC = Blank }
```

```
procedure STARTKATA      { mengabaikan satu atau beberapa Blank }
{ I.S. : CC sembarang  }
{ F.S. : LKata = 0, dan CC = Mark
         LKata ≠ 0, LKata adalah panjang kata yang sudah diakuisisi,
         CC karakter pertama  sesudah karakter terakhir kata }
```

KAMUS LOKAL

ALGORITMA

```
    START
    Ignore_Blank
    HITUNGPANJANG
```

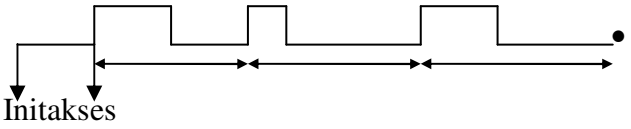
```
procedure ADVKATA
{ I.S. : LKata ≠ 0; CC adalah karakter sesudah karakter terakhir kata yg
         sudah diakuisisi  }
{ F.S. : LKata = 0, dan CC = Mark
         atau LKata ≠ 0, LKata adalah panjang kata yang sudah diakuisisi,
         CC karakter pertama  sesudah karakter terakhir kata  }
```

KAMUS LOKAL

ALGORITMA

```
    Ignore_Blank
    HITUNGPANJANG
```

Versi 3:
Mengabaikan Blank pada awal pita, dan memproses sisanya. Versi ini menggunakan model akuisisi kata yang berbeda dengan versi1 dan versi 2, yaitu model akuisisi kata TANPA MARK, artinya kata yang diakuisisi tidak pernah merupakan kata 'kosong'. Akuisisi kata dimulai dengan karakter pertama dari suatu kata sampai dengan karakter pertama dari kata berikutnya (atau titik jika merupakan kata terkahir). Model akuisisi ini mengharuskan adanya suatu prosedur INITAKSES, yang memposisikan CC pada karakter pertama kata pertama (berarti mengabaikan Blank sebelum karakter pertama yang ada pada pita).




```
Program PANJANGRATAKATA3
{ Versi 3 : SKEMA PEMROSESAN tanpa Mark }

KAMUS
    constant Mark : character = '.'
    constant Blank : character = ' '
    LKata : integer    { panjang kata terakhir yang susah diakuisisi }
    NbKata : integer   { banyaknya kata pada pita }
    LTotal : integer   { akumulasi panjang kata }

    procedure Ignore_Blank   { mengabaikan satu atau beberapa Blank }
    { I.S. : CC adalah sembarang  }
    { F.S. : CC ≠ Blank, atau CC = Mark }
    procedure HITUNGPANJANG  { menghitung panjang kata }
    { I.S. : CC adalah karakter pertama dari kata }
    { F.S. : CC = Blank, atau  CC = Mark;  CC adalah karakter sesudah huruf
    terakhir kata yang diakuisisi; LKata bebisi panjang kata yang sudah
    diakuisisi }
    procedure INITAKSES      { mengabaikan satu atau beberapa Blank pada awal
                               pita}
    { I.S. : CC sembarang }
    { F.S. : CC = Mark; atau CC = karakter pertama dari kata yang akan
    diakuisisi }
    procedure ADVKATA
    { I.S. : CC adalah karakter pertama  kata yg akan diakuisisi  }
    { F.S. : LKata adalah panjang kata yang sudah diakuisisi, CC karakter }
    {        pertama kata yang berikutnya, mungkin Mark }

ALGORITMA
    INITAKSES
    LTotal ← 0
    NbKata ← 0
    while (CC ≠ Mark)  do
        ADVKata
        LTotal ← LTotal + LKata
        NbKata ← NbKata + 1
    depend on NbKata
        NbKata ≠ 0 : output (LTotal/NbKata)
        NbKata = 0 : output ("Pita tidak mengandung kata")
```

Penjelasan:
Pada skema di atas, yang menggunakan WHILE, ADVKata dilakukan sebelum proses (bukannya sesudah proses). Hal ini disebabkan karena akuisisi kata dilakukan di dalam badan pengulangan.

```
procedure Ignore_Blank       { mengabaikan satu atau beberapa Blank }
{ I.S. : CC adalah sembarang }
{ F.S. : CC ≠ Blank, atau CC = Mark }
```

KAMUS LOKAL

ALGORITMA

```
    { I.S. : CC sembarang }
    while   (CC = Blank) and (CC ≠ Mark) do
        ADV
    { F.S. : CC ≠ Blank or CC = Mark }
```

```
procedure HITUNGPANJANG      { menghitung panjang kata }
{ I.S. : CC adalah karakter pertama dari kata, CC ≠ Mark}
{ F.S. : CC = Blank atau CC = Mark; CC adalah karakter sesudah huruf
terakhir kata yang diakuisisi; LKata berisi panjang kata yang sudah
diakuisisi }
```

KAMUS LOKAL

ALGORITMA

```
    LKata ← 1         { CC adalah karakter pertama pita! }
    iterate
        ADV
    stop (CC = Mark) or (CC = Blank)
        LKata ← LKata + 1
    { CC = Mark or CC = Blank }
```

```
procedure INITAKSES
{ mengabaikan satu atau beberapa Blank pada awal pita}
{ I.S. : CC sembarang }
{ F.S. : CC = Mark; atau CC = karakter pertama dari kata yang akan
diakuisisi }
```

KAMUS LOAKL

ALGORITMA

```
    START
    Ignore_Blank
```

```
procedure ADVKATA
{ I.S. :  CC adalah karakter pertama kata yg akan diakuisisi }
{ F.S. :  LKata adalah panjang kata yang sudah diakuisisi, CC karakter
pertama kata yang berikutnya, mungkin Mark }
```

KAMUS LOKAL

ALGORITMA

```
    HITUNGPANJANG
    Ignore_Blank
```

Latihan Soal

HURUF HIDUP

Diberikan sebuah mesin karakter dengan pita berisi karakter (mungkin kosong), hitunglah:

- banyaknya kemunculan huruf hidup yang muncul pada pita tersebut.
- frekuensi huruf hidup.
- banyaknya kemunculan setiap huruf hidup.

Definisikanlah dengan jelas apa yang dimaksud dengan huruf hidup.

HURUF MATI

Diberikan sebuah mesin karakter dengan pita berisi karakter (mungkin kosong) hitunglah:

- banyaknya kemunculan huruf mati yang muncul pada pita tersebut.
- frekuensi huruf mati.
- banyaknya kemunculan setiap huruf mati.

Definisikanlah dengan jelas apa yang dimaksud dengan huruf mati.

HURUF HIDUP muncul lebih sedikit dari Huruf MATI

Diberikan sebuah mesin karakter dengan pita berisi karakter (mungkin kosong) periksalah apakah banyaknya huruf hidup yang muncul pada pita tersebut lebih sedikit daripada huruf mati yang muncul. Definisikanlah dengan jelas apa yang dimaksud dengan huruf hidup.

HITUNG KATA

Diberikan sebuah mesin karakter dengan pita berisi karakter (mungkin kosong). Buatlah algoritma untuk menghitung banyaknya KATA yang ada pada pita tersebut. Mula-mula, Anda harus mendefinisikan apa itu KATA.

Definisi: kata adalah pasangan:

C1, C2, dengan C1 = sembarang huruf dan C2 = ' ', atau

C1, C2, dengan C1 = sembarang huruf dan C2 = '.'

Selanjutnya, tuliskanlah algoritmanya berdasarkan ide ini dari contoh-contoh di atas.

KATA BERAKHIR 'S'

Diberikan sebuah mesin karakter dengan pita berisi karakter (mungkin kosong), Buatlah algoritma untuk menghitung banyaknya KATA yang diakhiri 'S', yang ada pada pita tersebut.

Definisi kata yang diakhiri S adalah pasangan yang terdiri dari 3 karakter:

C1, C2, C3 dengan C1 = sembarang, C2 = S dan C3 = ' ', atau

C1, C2, C3 dengan C1 = sembarang, C2 = 'S' dan C3 '.'

Selanjutnya, tuliskanlah algoritmanya berdasarkan ide ini dari contoh-contoh di atas.

SUPPRESSBLANK

Diberikan sebuah mesin karakter dengan pita berisi karakter (mungkin kosong): Buatlah algoritma untuk menghilangkan BLANK yang berlebihan. BLANK yang berlebihan adalah:

- satu atau lebih BLANK sebelum huruf pertama yang bukan BLANK.
- satu atau lebih BLANK sesudah huruf terakhir yang bukan BLANK.
- lebih dari satu BLANK di antara dua buah kata.

KONVERSI

a. Diberikan sebuah mesin karakter dengan pita berisi karakter (mungkin kosong) dan diakhiri titik, setiap karakter hanya mengandung karakter angka '0' dan '1' sehingga mewakili suatu bilangan biner. Tuliskanlah algoritma untuk melakukan konversi deretan karakter biner yang muncul pada pita menjadi suatu besaran integer dalam desimal.
b. Diberikan sebuah mesin karakter dengan pita berisi karakter (mungkin kosong) dan diakhiri titik, setiap karakter hanya mengandung karakter angka '0'..'9'. Tuliskanlah algoritma untuk melakukan konversi deretan karakter yang muncul pada pita menjadi suatu besaran integer.
c. Diberikan sebuah mesin karakter dengan pita berisi karakter (mungkin kosong) dan diakhiri titik, setiap karakter hanya mengandung karakter angka '0'..'9' dan **koma**. Tuliskanlah algoritma untuk melakukan konversi deretan karakter yang muncul pada pita menjadi suatu besaran real. Contoh : 34,5.

EKSPRESI ARITMATIKA

a. Pada persoalan ekspresi aritmatika pada contoh di atas, ingin pula disertakan prosedur penanganan kesalahan. Kesalahan apa saja yang mungkin terjadi, dan bagaimana program Anda menanganinya? Buatlah spesifikasinya.
b. Pada ekspresi di atas, operator dan operan bukan hanya terdiri dari satu karakter. Operan boleh berupa NAMA (deretan karakter) dan setiap nama mempunyai nilai. Ketika perhitungan ekspresi, yang dipakai adalah nilainya. Buatlah spesifikasi dan modifikasilah program evaluasi ekspresi aritmatika yang diberikan.

Latihan soal menggunakan mesin karakter dan mesin kata, untuk memroses teks karakter demi karakter:

1. Buatlah sebuah program yang menghitung banyaknya kemunculan kata ketiga (jika ada).
2. Buatlah sebuah program yang mencetak kata terpanjang dan yang terpendek serta kemunculannya.
3. Buatlah sebuah program yang membuat statistik seperti yang dibuat oleh MS Word: *pages*, *characters* (*no spaces*), characters (*including spaces*), *Words*, *lines*, *paragraph*.
4. Buatlah browser sederhana (yang menampilkan teks dan mengolah "tag").

Latihan soal menggunakan mesin karakter dan mesin kata, untuk memroses *source code*:

1. Buatlah sebuah program yang memeriksa apakah blok dalam sebuah program Pascal seimbang (beres).
2. Buatlah sebuah program yang memeriksa apakah blok dalam sebuah program C seimbang (beres).
3. Buatlah sebuah program yang mencetak semua nama type, nama konstanta dan nama variabel yang muncul dalam sebuah program bahasa Pascal yang hanya mengandung program utama, dan programmernya disiplin untuk menuliskan semua variabel setelah kata VAR, semua type setelah kata TYPE dan semua konstanta setelah kata CONST.
4. Prasyarat untuk membuat soal ini: Anda harus mempunyai pemahaman yang "sedikit" tentang "tag" HTML:
   - Buatlah sebuah program yang menerima input sebuah file HTML dan menampilkan teksnya tanpa tag (membuang semua tag).
   - Buatlah sebuah program yang mencek apakah sebuah file HTML adalah file HTML yang valid (tag-nya seimbang).
   - Buatlah daftar tag yang muncul dalam sebuah file HTML.

Studi Kasus Mesin Karakter dan Tabel

Hitung While

Diberikan suatu pita katakter yang hanya mengandung abjad, blank, dan diakhiri titik, harus dicari kemunculan kata 'while' pada pita tersebut.
Didefinisikan sebuah type **kata** : < TabKata : array [1..50] of character ,
Length : integer >
dan sebuah fungsi **KataSama** yang menerima masukan Kata1 dan Kata2 yang bertype kata dan mengirimkan 'true' jika kedua kata sama, atau mengirimkan false jika kedua kata tidak sama.

Solusi yang diberikan di bawah ini memakai model akuisisi kata tanpa mark, seperti pada Versi 3 soal menghitung panjang kata rata-rata pada subbab Mesin Karakter. Prosedur menghitung panjang kata diadaptasi untuk mengakuisisi kata.

```
Program HITUNGWHILE
{ Menghitung kemunculan 'WHILE' pada suatu pita karakter }

KAMUS
    constant Mark  : character = '.'
    constant Blank : character = ' '
    constant NMax  : integer = 50 { jumlah maksimum karakter pada satu kata }
    type Kata : < TabKata : array [1..NMax] of character,
                  Length  : integer >
    KataWHILE : Kata      { kata yang menyimpan 'while' }
    CurrentKata : Kata    { kata terakhir sudah diakuisisi,
                            siap dibandingkan dg Kata WHILE }
    NWhile : integer      { banyaknya 'while' pada pita }

    procedure Ignore_Blank   { mengabaikan satu atau beberapa Blank }
    { I.S. : CC adalah sembarang }
    { F.S. : CC ≠ Blank, atau CC = Mark }

    procedure INITAKSES      { mengabaikan satu atau beberapa Blank pada awal
                               pita }
    { I.S. : CC sembarang }
    { F.S. : CC = Mark; atau CC = karakter pertama dari kata yang akan
             diakuisisi }

    procedure ADVKATA (output CKata : Kata)
    { I.S. : CC adalah karakter pertama  kata yg akan diakuisisi }
    { F.S. : CKata adalah kata terakhir yang sudah diakuisisi, CC karakter }
    {        pertama dari  kata yang berikutnya, mungkin Mark }

    function KataSama (Kata1, Kata2 : Kata) → boolean
    { True jika Kata1 sama dengan Kata2, false jika tidak }

ALGORITMA
    { Inisialisasi kata 'while ' }
    KataWhile.Length ← 5
    KataWhile.TabKata[1] ← 'w'
    KataWhile.TabKata[2] ← 'h'
    KataWhile.TabKata[3] ← 'i'
    KataWhile.TabKata[4] ← 'l'
    KataWhile.TabKata[5] ← 'e'
    NWhile ← 0
    INITAKSES
    while (CC ≠ Mark)  do
        ADVKata(CurrentKata)
        if (KataSama(KataWhile,CurrentKata)) then
            NWhile ← NWhile + 1
    output (NWhile)
```

```
procedure INITAKSES
{ Mengabaikan satu atau beberapa Blank pada awal pita }
{ I.S. : CC sembarang  }
{ F.S. : CC = Mark; atau CC = karakter pertama dari kata yang akan
         diakuisisi }

KAMUS LOKAL

ALGORITMA
    START
    Ignore_Blank
```

```
procedure Ignore_Blank   { mengabaikan satu atau beberapa Blank }
{ I.S. : CC adalah sembarang }
{ F.S. : CC ≠ Blank, atau CC = Mark }
```

```
KAMUS LOKAL
```

```
ALGORITMA
    { I.S. : CC sembarang }
    while (CC = Blank) and (CC ≠ Mark) do
        ADV
    { F.S. : CC ≠ Blank or CC = Mark }
```

```
procedure ADVKATA (output CKata : Kata)
{ I.S. : CC adalah karakter pertama kata yg akan diakuisisi }
{ F.S. : CKata adalah kata terakhir yang sudah diakuisisi, CC adalah
karakter pertama dari  kata yang berikutnya, mungkin Mark }
```

```
KAMUS LOKAL
    procedure SALINKATA (output CKata : Kata)
    { mengakuisisi kata, menyimpan dalam CKata }
    { I.S. : CC adalah karakter pertama dari kata }
    { F.S. : CC = Blank, atau  CC = Mark;  CC adalah karakter sesudah huruf
             terakhir kata yang diakuisisi; CKata berisi kata yang sudah
             diakuisisi }
```

```
ALGORITMA
    SalinKata(CKata)
    Ignore_Blank
```

```
procedure SALINKATA (output CKata : Kata)
{ mengakuisisi kata, menyimpan dalam CKata }
{ I.S. : CC adalah karakter pertama dari kata }
{ F.S. : CC = Blank, atau CC = Mark;  CC adalah karakter sesudah huruf
terakhir kata yang diakuisisi; CKata berisi kata yang sudah diakuisisi }
```

```
KAMUS LOKAL
    i : integer
```

```
ALGORITMA
    i ← 1
    iterate
        CKata.TabKata_i ←  CC
        ADV
    stop (CC = Mark) or (CC = Blank)
        i ← i + 1
    { CC = Mark or CC = Blank }
    CKata.Length ← i
```

Catatan :

Pada solusi di atas, jika panjang kata melebihi ukuran tabel, maka program akan salah. Untuk menangani hal ini, algoritma harus dikoreksi.

```
procedure SALINKATA1 (output CKata : kata)
{ mengakuisisi kata, menyimpan dalam CKata }
{ I.S. : CC adalah karakter pertama dari kata }
{ F.S. : CC = Blank, atau CC = Mark; CC adalah karakter sesudah huruf
terakhir kata yang diakuisisi; CKata berisi  kata yang sudah diakuisisi,
Jika banyaknya karakter melebihi Nmax, "sisa" kata dibuang }
```

```
KAMUS LOKAL
    i : integer
```

```
ALGORITMA
    i ← 1
    iterate
        CKata.TabKata_i ←  CC
        ADV
    stop (CC = Mark) or (CC = Blank) or (i = NMax)
        i ← i + 1
    { CC = Mark or CC = Blank or i = NMax }
    CKata.Length ← i
    if (i = NMax) then    { abaikan sisa kata }
        while (CC ≠ Blank) and (CC ≠ Mark) do
            ADV
        { CC = Mark or CC = Blank }
```

```
function KataSama (Kata1, Kata2 : Kata) → boolean
{ Menghasilkan true jika Kata1 sama dengan Kata2, false jika tidak }
{ Mencari huruf pada posisi yang sama, yang berbeda }
```

```
KAMUS LOKAL
    i : integer
```

```
ALGORITMA
    depend on (Kata1.Length, Kata2.Length)
        Kata1.Length ≠ Kata2.Length : →  false
        Kata1.Length = Kata2.Length :
            i ← 1
            while (i < Kata1.Length) and (Kata1.TabKata_i = Kata2.TabKata_i) do
                i ← i + 1
            { i = Kata1.Length or Kata1.TabKata_i ≠ Kata2.TabKata_i }
            → (Kata1.TabKata_i = Kata2.TabKata_i)
```

Palindrom

Tuliskanlah sebuah fungsi yang menerima masukan sebuah kata yang direpresentasi dalam array, memeriksa apakah kata itu PALINDROM dan mengirimkan sebuah harga boolean (true jika kata tsb. palindrom, false jika tidak. Palindrom artinya kata yang jika dibaca dari kiri ke kanan atau kanan ke kiri sama saja.

Contoh Palindrom : ANA
NABABAN
KASUR RUSAK
KASUR NABABAN RUSAK

```
function Palindrom (K : Kata) → boolean
{ True jika K adalah palindrom }

KAMUS LOKAL
   i, j : integer

ALGORITMA
   i ← 1
   j ← K.Length
   while (i < j) and (K.TabKata_i = K.TabKata_i) do
      i ← i + 1
      j ← j - 1
   { i = j or K.TabKata_i ≠ K.TabKata_i }
   →  (K.TabKata_i = K.TabKata_j)
```

Catatan:

1. Perhatikan bahwa persoalan Palindrom dapat diselesaikan dengan yang pernah kita lakukan: salin kata ke sebuah kata sementara, kemudian pakai fungsi KataSama untuk memeriksa kata asal dengan kata yang tersimpan pada kata temporer.
2. Solusi yang diambil pada algoritma di atas adalah solusi dengan menggunakan sebuah kata "logik" yang mempunyai urutan akses "mundur".

Latihan Soal

1. SKIPSEPARATOR

Semua prosedur Ignore_blank yang dituliskan berlaku untuk mengabaikan Blank, dan hanya akan dipakai jika pemisah antara dua buah kata adalah Blank. Buatlah prosedur yang mampu mengabaikan semua jenis separator yang memisahkan kata, misalnya selain Blank adalah koma, titi koma, dsb.

2. INVERSE

Dibaca sebuah pita dari mesin karakter yang diakhiri titik dan maksimum terdiri dari 50 karakter (termasuk titik). Karakter yang dibaca itu harus dituliskan kembali dengan urutan terbalik. Tuliskanlah algoritmanya

Contoh : Jika dibaca AKU ANAK DESA. maka akan ditulis ASED KANA UKA.

3. FREKUENSI KATA PERTAMA

Dibaca sebuah pita dari mesin karakter yang diakhiri titik. Hitunglah frekuensi kemunculan kata pertama dalam pita tersebut. Catatan: definisikan dulu apa yang dimaksud dengan kata. Andaikata teks selalu dalam bahasa Indonesia, dan kata terpanjang dalam bahasa Indonesia terdiri dari 35 karakter. Contoh: aku pergi ke pasar kemudian aku pulang ke rumah supaya aku dapat mandi. Output: Frekuensi 'aku' adalah 3/12.

4. ANAGRAM

Tuliskanlah sebuah fungsi yang menerima masukan dua buah kata yang direpresentasi dalam array, memeriksa apakah kedua kata itu ANAGRAM dan mengirimkan sebuah harga boolean (true jika kedua kata anagram, false jika tidak). Kata dikatakan Anagram jika terdiri dari huruf yang sama.

Contoh : SEBAB dan BEBAS adalah anagram
BAGUS dan GABUS adalah anagram
SUPER dan PUSER adalah anagram.

5. HITUNG WHILE

Tuliskan kembali algoritma menghitung 'while' jika pita karakter adalah sebuah sorce program dalam bahasa Pascal yang **tidak mengandung kesalahan sintaks** (teks mengandung karakter-karakter lain selain abjad, dan separator antara 'kata' bukan hanya blank!). Bagaimana jika program mengandung kesalahan sintaks?

6. TELEGRAM

Diberikan sebuah pita karakter yang diakhiri titik yang berisi telegram, yaitu deretan abjad berupa kata atau kata yang mewakili tanda baca, dipisahkan blank dan diakhiri dengan kata STOP, harus dihitung tarif telegram. Tarif telegram ditentukan oleh banyaknya kata yang muncul, sedangkan kata yang mewakili tanda baca tidak dihitung.

Contoh :

Teks	Jumlah Kata
SEGERA PULANG KOMA NENEK SAKIT TITIK STOP.	4
APA KABAR SAYA SENANG STOP.	4
HAI STOP.	1

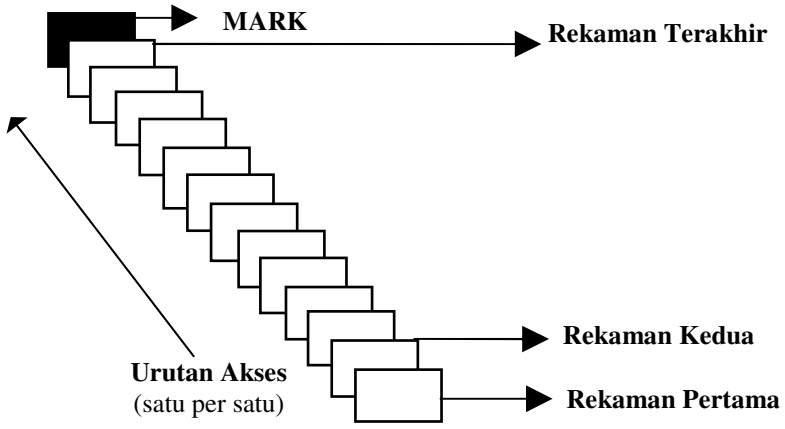
SEQUENTIAL FILE

Definisi

Sequential file (arsip sekuensial) adalah sekumpulan rekaman yang disimpan dalam media penyimpanan sekunder komputer, yang dapat diakses secara sekuensial **mulai dari rekaman pertama sampai dengan rekaman yang terakhir, rekaman per rekaman** secara searah. Rekaman terakhir adalah rekaman fiktif, yang menandai akhir dari arsip. Pada beberapa implementasi, rekaman fiktif ini disebut sebagai EOF (*End Of File*). Arsip sekuensial berasal dari hasil perekaman (penulisan) yang juga dilakukan rekaman per rekaman.

Setiap rekaman boleh berisi type dasar ataupun type terstruktur yang telah didefinisikan, setiap rekaman strukturnya sama. Elemen dari rekaman disebut sebagai "field", ada field (atau juga sekumpulan field) rekaman yang disebut sebagai "*key*" karena kekhususannya dalam proses. Jika *key* dari setiap rekaman tidak ada yang sama (unik), maka key menjadi identitas rekaman, dan disebut "*primary key*".

Setiap rekaman dapat diakses dan dikonsultasi (dibaca) menurut urutannya dengan pemanggilan primitif akses yang tersedia. Perekaman dapat dilakukan melalui primitif penulisan. Perhatikan bahwa suatu arsip sekuensial hanya dapat disiapkan hanya pada salah satu modus operasi: diakses/dibaca, **atau** ditulis. Dalam definisi arsip sekuensial, tidak pernah sebuah arsip diproses untuk kedua modus: dibaca dan sekaligus ditulis. Primitif-primitif tsb., akan didefinisikan kemudian.



Cara pendefinisian:

```
type rekaman : <...> { sebuah type terdefinisi untuk setiap rekaman }
NamaArsip : SEQFILE of
            (*) <nama_rek> : rekaman
            (1) <mark>
```

Dengan catatan bahwa (*) mungkin kosong, 1 rekaman atau lebih.
Domain setiap rekaman **:** sesuai dengan domain masing-masing rekaman
Konstanta : sebuah rekaman

Primitif akses untuk arsip sekuensial :

```
procedure ASSIGN (input NamaArsip, NamaFisik)
{ Arsip sekuensial yang namamnya dikenal di dalam program sebagai NamaArsip,
secara fisik diberi nama NamaFisik
  I.S. : sembarang
  F.S. : Arsip dengan NamaArsip pada program siap dipakai
}
```

Catatan:

Pada beberapa pemroses bahasa, primitif ASSIGN diberikan dalam teks algoritma, dan dalam beberapa pemroses bahasa pemrograman yang lain diberikan di luar program (pada *Job Control Language*). Pada teks algoritma di buku ini, perintah ASSIGN tidak diberikan di dalam teks algoritma.

```
procedure OPEN (input NamaArsip, <rekaman>)
{ Arsip sekuensial siap dibaca  Rekaman pertama yang informasinya ada
  pada <rekaman> dapat diakses.
  I.S. : sembarang
  F.S. : informasi pada rekaman pertama siap diakses, dengan mengacu
         kepada <rekaman>
}
```

```
procedure READ (input NamaArsip, <rekaman>)
{ Rekaman sesudah rekaman yang sedang "Current", yang dapat diakses.
  I.S. : <rekaman> bukan merupakan mark, sebut sebagai
         Current_Rekaman
  F.S. : Arsip dimajukan satu rekaman, <rekaman> berisi informasi
         yang disimpan pada rekaman sesudah Current_Rekaman.
         Mungkin <rekaman> yang baru adalah mark
}
```

```
procedure CLOSE (input NamaArsip)
{ Arsip sekuensial "ditutup", tidak dapat diakses maupun ditulisi
  lagi.
  I.S. : sembarang
  F.S. : Arsip tidak dapat diproses lagi
}
```

Primitif perekaman untuk arsip sekuensial:

```
procedure REWRITE (input/output NamaArsip)
{ Arsip sekuensial siap untuk direkam.
  I.S. : sembarang
  F.S. : Arsip sekuensial yang bernama NamaArsip siap untuk direkam
         pada posisi pertamanya
}
```

```
procedure WRITE (input/output NamaArsip, <rekaman>)
{ Data  pada <rekaman> direkam pada posisi aktual arsip. Kemudian
  posisi diajukan satu.
  I.S. : arsip sekuensial  berada pada posisi yang telah siap
         menerima rekaman, <rekaman> bukan merupakan MARK.
  F.S. : <rekaman> direkam pada posisi yang telah disiapkan, arsip
         diajukan satu posisi.
        Jika <rekaman> yang diisikan ke arsip adalah elemen fiktif
         yang dimaksudkan sebagai MARK, maka arsip tak dapat lagi
         direkami. }
```

Contoh 1:

Sebuah arsip sekuensial berisi data mahasiswa, yang setiap rekamannya memuat data NIM, Nama dan Nilai akhir mahasiswa.. Maka dapat dituliskan :

```
type rekaman : < NIM : integer,
                 Nama : string,
                 Nilai : integer [0..100] >
ArsipMhs : SEQFILE of
            (*) RekMhs : rekaman
            (1) <9999999, "", 0>
```

Domain setiap rekaman : sesuai dengan domain masing-masing rekaman.
Konstanta : sebuah rekaman, misalnya:
`<7473001,"Juliette",95>` `<8690022,"Laura",80>`

Cara akses : rekaman pertama : `OPEN (ArsipMhs,RekMhs)`
Cara akses : NIM ≠ 9999999 : `READ (ArsipMhs,RekMhs )`

Cara menyiapkan untuk direkam : `REWRITE (ArsipMhs)`
Cara mengisi :
```
WRITE (ArsipMhs,RekMhs) { harga current }
WRITE (ArsipMhs,<7473002,"Davy Rindt",96>) { konstanta }
WRITE (ArsipMhs, Rek1) { Rek1 ber-type rekaman }
```

Cara mengisi akhir rekaman : `WRITE (ArsipMhs,<9999999,"",0>)  { Mark }`

Contoh 2 :

Sebuah arsip sekuensial berisi teks, maka setiap rekamannya adalah satu karakter. Misalnya MARK adalah `'#'`:

```
type rekaman : character
Dokumen : SEQFILE of
            (*) CC : rekaman
            (1) <'#'>
```

Domain setiap rekaman : `character`
Konstanta : sebuah rekaman, misalnya:
`<'A'>` `<'0'>` `<'#'>`
Cara akses : rekaman pertama : `OPEN (Dokumen, CC)`
Cara akses : `CC ≠ '#'` : `READ (Dokumen, CC)`
Cara menyiapkan untuk direkam : `REWRITE (Dokumen)`
Cara mengisi :
```
WRITE (Dokumen,CC)     { harga current }
WRITE (Dokumen,<'A'>) { konstanta }
WRITE (Dokumen,Kar)    { dari nama lain, Kar bertype rekaman }
```
Cara mengisi akhir rekaman : `WRITE (Dokumen, <'#'>)`

Pemrosesan Sebuah Arsip Sekuensial

Jika setiap rekaman harus diproses dengan cara sama, pemrosesan arsip sekuensial dapat dilakukan dengan memakai skema pemrosesan sekuensial dengan mark.

Contoh:

Dibaca sebuah arsip sekuensial bernama

```
type rekaman : < NIM : integer, nilai : integer [0..100] >
ArsipMhs : SEQFILE of
                (*) RekMhs : rekaman
                (1) <9999999, 99>
```

Analisis : pemrosesan sekuensial dari elemen arsip sekuensial.

Model tanpa MARK, jika i adalah deret yang diproses, i bernilai 1, 2, 3, ..., N

EOP adalah NIM = 9999999

First_Elmt : `OPEN (ArsipMhs, RekMhs)`

Next_Elmt : `READ (ArsipMhs, RekMhs)`

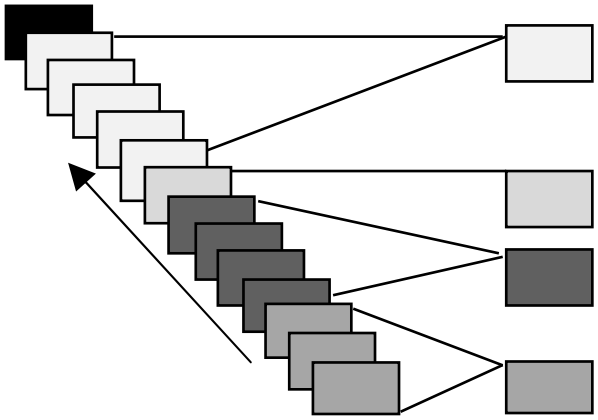
Proses : membaca arsip sambil menghitung nilai rata-rata mahasiswa.

```
Program NILAIRATA_RATA
{ Model proses sekuensial dengan mark,
  dengan penanganan kasus kosong }

KAMUS
   type rekaman : < NIM : integer, nilai : integer[0..100] >
   ArsipMhs : SEQFILE of
               (*) RekMhs : rekaman { setiap mahasiswa punya 1 rekaman }
               (1) <9999999, 99>
   SumNil : integer  { jumlah nilai }
   JumMhs : integer  { jumlah mahasiswa }

ALGORITMA
   OPEN(ArsipMhs, RekMhs)               { First_Elmt }
   if (RekMhs.NIM = 9999999) then
      output ("Arsip kosong")
   else
      SumNil ← 0; JumMhs ← 0            { Inisialisasi }
      repeat
         SumNil ← Sumnil + RekMhs.nilai; JumMhs ← JumMhs + 1     { Proses }
         READ(ArsipMhs,RekMhs)          { Next_Elmt }
      until (RekMhs.NIM = 9999999)      { EOP }
      output (Sum/JumMhs)               { Terminasi }
   CLOSE(ArsipMhs)
```

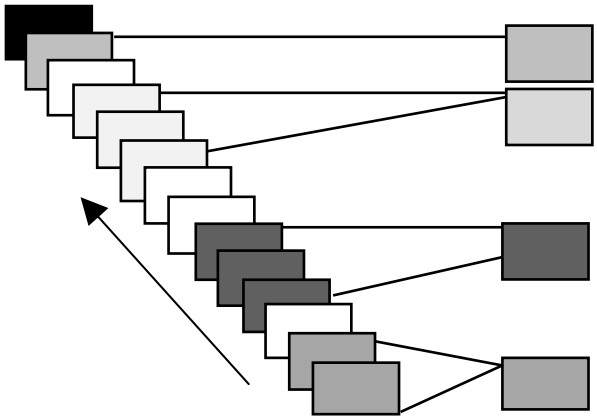
Algoritma Konsolidasi

Didefinisikan sebuah *sequential file* yang terurut, arsip tersebut mengandung kelompok-kelompok data dengan kunci sama yang harus diproses sebagai satu kesatuan. Ada dua model arsip semacam ini:

1. Tanpa separator, artinya kita mengenali adanya kelompok yang lain karena kunci berubah.



2. Dengan separator, artinya ada rekaman tertentu yang memisahkan satu kelompok dan kelompok lainnya. Separator ini boleh satu rekaman atau lebih dari satu rekaman. Pada contoh berikut, separator adalah "kartu putih".



Berikut ini diberikan algoritma standar untuk konsolidasi. Dasarnya adalah pemakaian skema pemrosesan sekuensial dengan mark, yang setiap elemennya adalah satu kelompok.

Algoritma Konsolidasi Tanpa Separator

```
Program KONSOLIDASITanpaSeparator1
{ Tanpa penanganan kasus kosong }
{ Input : sebuah arsip sequential, terurut }
{ Proses :Mengelompokkan setiap kategori dan memrosesnya}
{ Output : Sesuai  hasil proses }
```

```
KAMUS
    { Keytype adalah suatu type dari kunci rekaman,
      Valtype adalah type dari harga rekaman }
    { Keytype dan Valtype harus terdefinisi
      Akhir arsip ditandai oleh mark dan Val }
    type rekaman : < KeyIn : keytype, { kunci }
                     ValIn :valtype   { harga lain yang direkam } >
    ArsipIn : SEQFILE of { input, terurut menurut kunci }
              (*) RekIn : rekaman
              (1) <mark, Val>
    EOP : boolean   { true jika rekaman yang dibaca kuncinya adalah mark }

    procedure Inisialisi_Seluruh_Categ   { Inisialisasi global }
    procedure Terminasi_Seluruh_Categ    { Terminasi global }
    Current_Categ : keytype              { Identifikasi kategori yang sedang
                                           diproses }
    procedure Proses_First_Elmt          { Inisialisasi sebuah kategori }
    procedure Init_Categ                 { Inisialisasi kategori }
    procedure Proses_Current_Categ       { Proses sebuah elemen dalam 1
                                           kategori }
    procedure Terminasi_Categ            { Terminasi sebuah kategori }
```

```
ALGORITMA
    Inisialisi_Seluruh_Categ
    OPEN(ArsipIn, RekIn)                 { First_Elmt }
    while not EOP do
        { Proses satu kategori }
        Init_Categ
        Current_Categ ← RekIn.KeyIn
        repeat
            Proses_Current_Categ
            READ(ArsipIn, RekIn)
        until (Current_Categ ≠ RekIn.KeyIn)
        { RekIn.KeyIn ≠ Current_Categ,
          RekIn.KeyIn adalah elemen pertama dari Next_Categ }
        Terminasi_Categ
    Terminasi_Seluruh_Categ
    CLOSE(ArsipIn)
```

```
Program KONSOLIDASITanpaSeparator2
{ Dengan penanganan kasus kosong }
{ Input : sebuah arsip sequential, terurut }
{ Proses :Mengelompokkan setiap kategori dan memrosesnya }
{ Output : Sesuai hasil proses }
```

```
KAMUS
   { Keytype adalah suatu type dari kunci rekaman,
     Valtype adalah type dari harga rekaman }
   { Keytype dan Valtype harus terdefinisi
     Akhir arsip ditandai oleh mark dan Val }
   type rekaman : < KeyIn : keytype,    { kunci }
                    ValIn : valtype     { harga lain yang direkam } >
   ArsipIn : SEQFILE of  { input, terurut menurut kunci }
             (*) RekIn : rekaman
             (1) <mark, Val>
   EOP : boolean   { true jika rekaman yang dibaca kuncinya adalah mark }

   procedure  Inisialisi_Seluruh_Categ  { Inisialisasi global }
   procedure  Terminasi_Seluruh_Categ   { Terminasi global }
   procedure  Kasus_Kosong              { Penanganan kasus kosong }
   Current_Categ : keytype              { Identifikasi kategori yang sedang
                                          diproses }
   procedure  Proses_First_Elmt         { Inisialisasi sebuah  kategori }
   procedure  Init_Categ                { Inisialisasi kategori }
   procedure  Proses_Current_Categ      { Proses sebuah elemen dalam 1
                                          Kategori }
   procedure  Terminasi_Categ           { Terminasi sebuah  kategori }
```

```
ALGORITMA
   OPEN(ArsipIn, RekIn)                 { First_Elmt }
   if (EOP) then
      Kasus_kosong
   else   { minimal ada satu kategori }
      Inisialisi_Seluruh_Categ
      repeat
         { Proses satu kategori }
         Init_Categ
         Current_Categ ← RekIn.KeyIn
         repeat
            Proses_Current_Categ
            READ(ArsipIn,RekIn)
         until (Current_Categ ≠ RekIn.KeyIn)
         { KeyIn ≠ Current_Categ, RekIn.KeyIn = elemen pertama Next_Categ }
         Terminasi_Categ
      until (EOP)
      Terminasi_Seluruh_Kategori
   CLOSE(ArsipIn)
```

Contoh Aplikasi 1:

Diketahui sebuah arsip nilai mahasiswa, satu mahasiswa dapat mempunyai beberapa buah nilai (karena dalam satu semester mengambil beberapa matakuliah dan setiap mahasiswa tidak sama matakuliahnya). Buat algoritma untuk menghitung nilai rata-rata setiap mahasiswa, dan membuat daftar nilai sederhana, yaitu menuliskan NIM dan nilai rata-rata setiap mahasiswa.

```
Program NILAIMAHASISWA
{ Input : sebuah arsip sekuensial berisi NIM dan nilai mahasiswa }
{ Proses : proses  setiap kategori adalah menghitung nilai rata-rata setiap
mahasiswa }
{ Output : NIM dan Nilai rata-rata setiap mahasiswa }
```

```
KAMUS
    type Keytype : integer
    type Valype  : integer [0..100]
    type rekaman : < NIM   : keytype,    { kunci }
                     Nilai : valtype     { nilai ujian } >
    ArsipMhs : SEQFILE of   { input, terurut menurut kunci }
                  (*) RekMhs : rekaman
                  (1) <9999999, 0>
    Current_NIM : integer   { identifikasi kategori yg sedang diproses }
    SumNil : integer        { Jumlah nilai seluruh matakuliah seorg mhs }
    NKuliah : integer       { Jumlah matakuliah seorang mahasiswa }
```

```
ALGORITMA
    { Inisialisasi : tidak ada }
    OPEN(ArsipMhs, RekMhs)                { First_Elmt }
    while (RekMhs.NIM ≠ 9999999) do       { not EOP }
        { Proses satu kategori = 1 NIM }
        SumNil ← 0                        { Init_Categ }
        NKuliah ← 0                       { Init_Categ }
        Current_NIM ← RekMhs.NIM
        repeat
            SumNil ← SumNil + RekMhs.Nilai
            NKuliah ← NKuliah + 1         { Proses_Current_Categ }
            READ(ArsipMhs, RekMhs)
        until (Current_NIM ≠ RekMhs.NIM)
        { NIM ≠ Current_NIM,
          NIM adalah elemen pertama dari Next_Categ }
        { Terminasi_Categ }
        output (Current_NIM, SumNil/NKuliah)
    CLOSE(ArsipMhs)
```

Contoh Aplikasi 2:

Idem Contoh Aplikasi 1, tetapi selain itu juga dikehendaki nilai rata-rata seluruh mahasiswa, jumlah nilai rata-rata setiap mahasiswa dibagi jumlah mahasiswa.

```
Program KONSOLIDASITanpaSeparator
{ Dengan penanganan kasus kosong }
{ Input : sebuah arsip sekuensial berisi NIM dan nilai mahasiswa }
{ Proses : Mengelompokkan setiap kategori dan memrosesnya : menghitung nilai
rata-rata setiap mahasiswa dan nilai rata-rata seluruh populasi mahasiswa }
{ Output : NIM Nilai rata-rata setiap mahasiswa, dan nilai rata-rata seluruh
mahasiswa }
```

```
KAMUS
   type Keytype : integer
   type Valype  : integer [0..100]
   type rekaman : < NIM   : keytype,    { kunci }
                    Nilai : valtype     { harga lain yang direkam } >
   ArsipMhs : SEQFILE of    { input, terurut menurut kunci }
              (*) RekMhs : rekaman
              (1) <9999999, 0>
   Current_NIM : integer    { identifikasi kategori yg sedang diproses }
   SumNil : integer         { Jumlah nilai seluruh matakuliah seorg mhs }
   NKuliah : integer        { Jumlah matakuliah seorang mahasiswa }
   NilRata : integer        { Nilai rata-rata seorang mahasiswa }
   SumNilTot : integer      { Jumlah nilai seluruh matakuliah seorg mhs }
   NMhs : integer           { Jumlah matakuliah seluruh mahasiswa }
```

```
ALGORITMA
   OPEN(ArsipMhs ,RekMhs)               { First_Elmt }

   if (RekMhs.NIM = 9999999) then       { EOP }
      output ("Arsip kosong")
   else

      repeat

         { Proses satu kategori = 1 NIM }
         SumNil ← 0; NKuliah ← 0      { Init_Categ }
         Current_NIM ← RekMhs.NIM
         repeat
            SumNil ← SumNil + RekMhs.Nilai   { Proses }
            NKuliah ← NKuliah + 1            { Proses_Current_Categ }
            READ(ArsipMhs, RekMhs)
         until (Current_NIM ≠ RekMhs.NIM)
         { NIM ≠ Current_NIM, RekMhs.NIM = elemen pertama Next_Categ }
         NilRata ← SumNil/NKuliah
         SumNilTot ← SumNilTot + NilRata
         NMhs ← NMhs + 1
         output (Current_NIM,NilRata)         { Terminasi_Categ }

      until (RekMhs.NIM = 9999999)            { EOP }

      output (SumNilTot/NMhs)                 { terminasi seluruh file }

   CLOSE (ArsipMhs)
```

Algoritma Konsolidasi Dengan Separator

Adanya kategori baru tidak dikenali dari *key*, tetapi dari separator.

```
Program KONSOLIDASIDenganSeparator
{ Input  : Sebuah arsip sequential }
{ Proses : Mengelompokkan setiap kategori dan memrosesnya }
{ Output : Sesuai hasil proses  }
```

```
KAMUS
    { Keytype adalah suatu type dari kunci rekaman,
      Valtype adalah type dari harga rekaman }
    { Keytype dan Valtype harus terdefinisi
      Akhir arsip ditandai oleh mark dan Val }
    type rekaman : < KeyIn : keytype,    { kunci }
                     ValIn : valtype     { harga lain yang direkam } >
    ArsipIn : SEQFILE of  { input, terurut menurut kunci }
                 (*) RekIn : rekaman
                 (1) <mark, Val>
    EOP : boolean      { True jika rekaman yang dibaca kuncinya adalah mark }
    procedure Inisialisi_Seluruh_Categ   { Inisialisasi global }
    procedure Terminasi_Seluruh_Categ    { Terminasi global }
    procedure Kasus_Kosong               { Penanganan kasus kosong }
    procedure Init_Categ                 { Inisialisasi untuk satu kategori }
    procedure Proses_Current_Categ       { Proses terhadap sebuah elemen
                                           Kategori }
    procedure Terminasi_Categ            { Terminasi sebuah  kategori }
    function Separator (K : Keytype) → boolean
    { True jika K adalah separator antar kategori }
```

```
ALGORITMA

    Inisialisasi_Seluruh_Categ
    OPEN(ArsipIn, RekIn)          { First_Elmt }
    if EOP then
       Kasus_Kosong
    else
       repeat

          { Skip separator }
          while not EOP and Separator(Keyin) do
             READ (ArsipIn, RekIn)      { KeyIn bukan  Separator,
                                          KeyIn adalah elemen pertama dari
                                          Next_Categ atau EOP }
             { Tidak dibedakan antara kasus kategori kosong dan tidak }
          { EOP atau not Separator}

          Init_Categ
          while (not EOP) and (not Separator(KeyIn)) do
             { Proses satu kategori }
             Proses_Current_Categ
             READ (ArsipIn, RekIn)
          { Separator atau EOP }
          Terminasi_Categ

       until EOP
    Terminasi_Seluruh_Categ
    CLOSE (ArsipIn)
```

Contoh Aplikasi:

Diberikan sebuah arsip teks yang dapat diakses *sequential* huruf per huruf. Hendak dihitung kata yang terpanjang dalam teks tersebut. Diandaikan bahwa teks hanya mengandung huruf dan "blank". Kata adalah sekumpulan huruf yang dipisahkan oleh satu atau beberapa blank.

```
Program KATATERPANJANG
{ Input  : sebuah arsip sequential, yang mewakili sebuah teks, }
{          setiap rekaman adalah sebuah karakter }
{ Proses : Menghitung kata terpanjang dalam teks }
{ Output : Panjang kata maksimum }
```

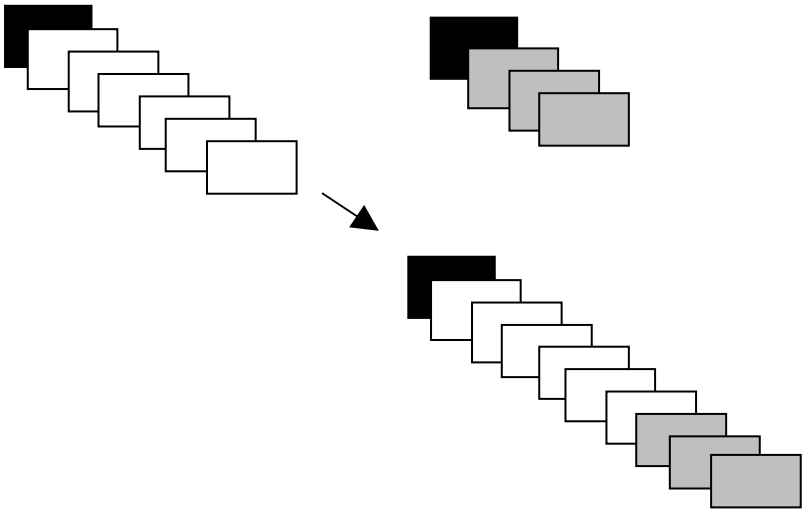
```
KAMUS
   { Keytype : type dari elemen arsip yg menentukan apakah elemen tsb.
               separator atau bukan}
   { Valtype adalah type dari harga rekaman, di sini tidak ada }
   KeyType : character
   constant mark  : character = '.'
   constant blank : character = ' '
   type rekaman : Keytype
   ArsipIn : SEQFILE of  { input, terurut menurut kunci }
             (*) CC : rekaman   { CC : sebuah karakter yang direkam }
             (1) <'.'>          { mark }
   PanjangKata : integer        { Panjang kata yang sedang dalam proses }
   MaxLength : integer          { Panjang kata maksimum }
   procedure Inisialisasi
   procedure Init_Categ         { Inisialisasi untuk satu kategori }
   procedure Proses_Current_Categ       { Proses terhadap sebuah elemen
                                          kategori }
   procedure Terminasi_Categ    { Terminasi sebuah  kategori }
   function Separator (K : Keytype) → boolean
   { True jika K adalah separator, di sini adalah blank : ' ' }
```

```
ALGORITMA
   OPEN(ArsipIn, CC)                          { First_Elmt }
   if (CC = mark) then
      output ("Arsip kosong")
   else
      Maxlength ← 0     { Diandaikan kata minimum terdiri dari 1 huruf }
      repeat
         { Skip separator, jika ada}
         while (CC ≠ mark) and (CC = blank) do
            { not EOP & Separator(Keyin) }
            READ (ArsipIn, CC)
            { CC bukan Separator, CC adalah huruf pertama dari sebuah kata
              atau Mark }
         { CC = mark or CC ≠ blank }
         PanjangKata ← 0
         while (CC ≠ mark) and (CC ≠ blank ) do
            { not Separator(KeyIn) and not EOP }
            PanjangKata ← PanjangKata + 1
            { Proses_Current_Categ }
            READ (ArsipIn, CC)
         { CC = blank or CC = mark }
         if (MaxLength < PanjangKata)  then
            MaxLength ← Panjang Kata         { Terminasi_Categ }
      until (CC = mark)                      { EOP }
      { Terminasi proses, Maxlength = 0 berarti Arsip hanya berisi blank }
      output (MaxLength)
   CLOSE(ArsipIn)
```

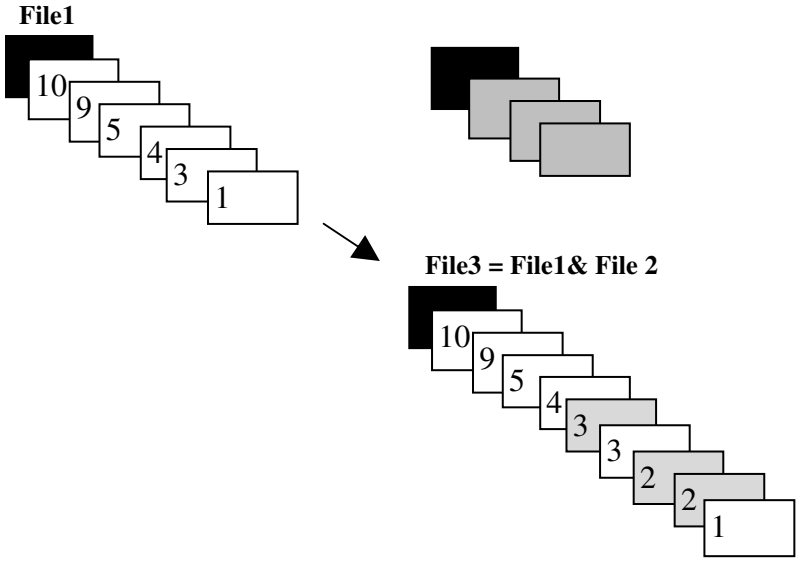
Algoritma Pemrosesan Dua Buah Arsip Sekuensial

Merging

Merging adalah penggabungan dua buah arsip. Yang paling sederhana adalah jika arsip yang pertama "dikonkatenasi" ke arsip kedua (artinya data dari arsip ke dua ditambahkan setelah rekaman terakhir arsip pertama dan membentuk arsip yang baru):



Cara di atas tak dapat dipakai jika kedua arsip sudah terurut, dan dikehendaki sebuah arsip hasil yang tetap terurut. Algoritma untuk penggabungan dua buah arsip terurut menjadi sebuah arsip yang terurut adalah:



```
Program MERGING1
{ Input  : Dua arsip sequential, terurut, sejenis }
{ Proses : Menggabung kedua arsip menjadi sebuah arsip yg terurut }
{          VERSI AND }
{ Output : Sequential file baru yang terurut }
```

```
KAMUS
   { Keytype adalah suatu type dari kunci rekaman,
     Valtype adalah type dari harga rekaman }
   { Keytype dan Valtype harus terdefinisi
     Akhir arsip ditandai oleh mark dan Val }
   type rekaman : < Key : keytype,      { kunci }
                    Val : valtype       { harga lain yang direkam } >
   ArsipIn1 : SEQFILE of    { input, terurut menurut kunci }
              (*) RekIn1 : rekaman
              (1) <mark, Val>
   ArsipIn2 : SEQFILE of    { input, terurut menurut kunci }
              (*) RekIn2 : rekaman
              (1) <mark, Val>
   ArsipOut : SEQFILE of    { output, terurut menurut kunci }
              (*) RekOut : rekaman
              (1) <mark, Val>
```

```
ALGORITMA
   OPEN(Arsip1, RekIn1)         { First_Elmt of Arsip1 }
   OPEN(Arsip2, RekIn2)         { First_Elmt of Arsip2 }
   REWRITE(ArsipOut)            { Menyiapkan arsip hasil : Arsip3 }
   while (RekIn1.Key ≠ mark) and (RekIn2.Key ≠ mark) do
      depend on (RekIn1.Key, RekIn2.Key)
         RekIn1.Key ≤ RekIn2.Key : WRITE(ArsipOut, RekIn1)
                                   READ(ArsipIn1, RekIn1)
         RekIn1.Key > RekIn2.Key : WRITE(ArsipOut, RekIn2)
                                   READ(ArsipIn2, RekIn2)
   { RekIn1.Key = mark or RekIn2.Key = mark }
   while (RekIn1.Key ≠ mark) do
      WRITE(ArsipOut, RekIn1)
      READ(ArsipIn1, RekIn1)
   { Akhir Arsip1, RekIn1.Key = mark }
   while (RekIn2.Key ≠ mark) do
      WRITE(ArsipOut, RekIn2)
      READ(ArsipIn2, RekIn2)
   { Akhir Arsip2, RekIn2.Key = mark }
   WRITE(ArsipOut, <mark,Val>)
   CLOSE(ArsipIn1)
   CLOSE(ArsipIn2)
   CLOSE(ArsipOut)
```

```
Program MERGING2
{ Input  : Dua arsip sequential, terurut menaik menurut kunci, sejenis, dan
           semua harga <Mark> }
{ Proses : Menggabung kedua arsip menjadi sebuah arsip yg terurut }
{          VERSI  OR }
{ Output : Sequential file baru yang terurut }
```

```
KAMUS
   { Keytype adalah suatu type dari kunci rekaman,
     Valtype adalah type dari harga rekaman }
   { Keytype dan Valtype harus terdefinisi
     Akhir arsip ditandai oleh mark dan Val }
   type rekaman : < Key : keytype,      { kunci }
                    Val : valtype       { harga lain yang direkam } >
   ArsipIn1 : SEQFILE of    { input, terurut menurut kunci }
                (*) RekIn1 : rekaman
                (1) <mark, Val>
   ArsipIn2 : SEQFILE of    { input, terurut menurut kunci }
                (*) RekIn2 : rekaman
                (1) <mark, Val>
   ArsipOut : SEQFILE of    { output, terurut menurut kunci }
                (*) RekOut : rekaman
                (1) <mark, Val>
```

```
ALGORITMA
   OPEN(ArsipIn1, RekIn1)        { First_Elmt of Arsip1 }
   OPEN(ArsipIn2, RekIn2)        { First_Elmt of Arsip2 }
   REWRITE(ArsipOut)             { Menyiapkan arsip hasil : Arsip3 }
   while (RekIn1.Key ≠ mark) or (RekIn2.Key ≠ mark) do
      depend on (RekIn1.Key, RekIn2.Key)
          RekIn1.Key ≤ RekIn2.Key : WRITE(ArsipOut, RekIn1)
                                    READ(ArsipIn1, RekIn1)
          RekIn1.Key > RekIn2.Key : WRITE(ArsipOut, RekIn2)
                                    READ(ArsipIn2, RekIn2)
   { RekIn1.Key = mark and RekIn2.Key = mark }
   WRITE(ArsipOut, <mark,Val>)
   CLOSE(ArsipIn1)
   CLOSE(ArsipIn2)
   CLOSE(ArsipOut)
```

Catatan :

1. **Versi-or** ini teks algoritmanya lebih singkat daripada **versi-and**.
2. Algoritma ini hanya benar jika mark adalah suatu nilai khusus yang dipakai untuk "menahan" maju ke rekaman berikutnya, sehingga arsip yang belum habis akan terproses sampai habis.
3. Versi ini tidak dapat digunakan secara umum karena kekhususan nilai mark tersebut. Bahkan tak dapat digunakan sama sekali jika mark adalah suatu nilai EOF yang ditentukan oleh sistem seperti pada kebanyakan sistem pengarsipan.

Updating dengan Transaction file

Updating adalah mengubah harga rekaman yang ada pada sebuah *master file* dengan data dari *transaction file*.

Berikut ini akan diberikan algoritma umum untuk meremajakan rekaman dari sebuah arsip sekuensial yang terurut dengan *key* unik (yang biasanya disebut sebagai Master File). Peremajaan dilakukan terhadap rekaman yang ada dilakukan berdasarkan arsip terurut lain (Update File), dengan *key* tidak unik. Artinya satu rekaman pada Master File dapat mengalami satu atau beberapa kali peremajaan. Hasil peremajaan dilakukan langsung terhadap arsip Master.

Catatan: Masalah harus dispesifikasikan dengan cermat. Bagaimana jika ada peremajaan terhadap master padahal *key* yang diremajakan tidak ada? Contoh: Peremajaan arsip saldo tabungan pada bank, dengan perjanjian jumlah pada arsip Update adalah negatif untuk pengambilan, positif untuk penabungan. Penabungan adalah dalam US $.

Rekaman Master adalah `<key : integer, Saldo : integer>`
Rekaman Update adalah `<key: integer, jumlah : integer>`

Arsip Master:	<1,23>	<3,34>	<6,200>	<16,10>	<22,50>	<30,0>	<999,0>
Arsip Update:	<3,2>	<3,4>	<16,-10>	<22,1>	<25, 50>	<30,5>	<999,0>
Arsip Master baru:	<1,23>	<3,40>	<6,200>	<16,0>	<22,51>	<30,5>	<999,0>

```
Program UPDATING
{ Input  : Dua arsip sequential (MasterFile dan Transaction File),
           terurut menaik menurut kunci yg sama }
{ Proses : Meremajakan sebuah field pada Master file berdasarkan Transaction
           File }
{ Output : Sequential file baru  yang terurut menaik menurut kunci }
{ Seq. processing terhadap Master file,
  Master File adalah pengendali pada loop luar }
```

```
KAMUS
   { Keytype : suatu type dari kunci rekaman,
     Akhir arsip ditandai oleh mark }
   type rekMaster : < KeyM : integer, Saldo : integer >
   type rekTrans  : < KeyT : integer, TransSaldo : integer >
   Master : SEQFILE of          { input,  terurut menurut kunci }
            (*) RekM : rekMaster
            (1) <9999,0>
   Transaction : SEQFILE of     { input, terurut menurut kunci }
                 (*) RekT : rekTrans
                 (1) <9999,0>
   NewMaster : SEQFILE of       { output, terurut menurut kunci }
                                { Master file baru yg telah diremajakan }
                 (*) rekNM : rekMaster
                 (1) <9999,0>
   Newsaldo : integer
```

```
ALGORITMA
   OPEN(Master, RekM)           { First_Elmt of Arsip Master }
   OPEN(Transaction, RekT)      { First_Elmt of Arsip Transaksi }
   REWRITE(NewMaster)           { Menyiapkan arsip hasil NewMaster }
   while (RekM.Key ≠ 9999) do
      while (RekT.Key < RekM.Key) and (RekT.Key ≠ 9999) do
         { Error, skip transaction file yg tidak ada pada master }
         READ(Transaction, RekT)
      { RekT.Key ≥ RekM.Key or RekT.Key = 9999 }
      if (RekT.Key = RekM.Key) then     { updating }
         { Init Update }
         NewSaldo ← RekM.Saldo
         repeat
            NewSaldo ← NewSaldo + RekT.TransSaldo
            READ(Transaction, RekT)
         until (RekT.Key ≠ RekM.Key) or (RekT.Key = 9999)
         { RekT.Key ≠ RekM.Key or RekT.Key = 9999 }
         { End update }
         WRITE(NewMaster, <RekM.Key,NewSaldo>)
      else   { RekT.Key > RekM.Key, tidak ada update thd MasterFile, salin }
         WRITE(NewMaster, <KeyM,Saldo>)
      READ(Master, RekM)        { Next Elmt of Master }
   { RekM.Key = 9999 }
   WRITE(NewMaster, <9999,0>)
   CLOSE(Master); CLOSE(Transaction)
   CLOSE(NewMaster)
```

Splitting

Splitting adalah pemecahan sebuah arsip menjadi dua atau lebih arsip. Algoritmanya tergantung pada kriteria pemecahannya.

Contoh :

- Memisahkan sebuah arsip pegawai menjadi beberapa arsip sesuai dengan kode golongan.
- Memisahkan arsip data percobaan sesuai dengan kriteria data (misalnya yang layak dipakai dan yang harus dibuang).

Latihan Soal

1. Tuliskanlah algoritma umum untuk menggabungkan dua buah arsip yang terurut menurut kunci tertentu, yang menghasilkan sebuah arsip terurut dengan key unik. Artinya jika ada kunci yang sama pada kedua arsip input, hanya salah satu rekaman yang ditulis ke arsip hasil. Rekaman yang "dibuang" karena duplikasi file harus dituliskan ke keluaran dengan perintah output.
   Catatan: Spesifikasikan dengan cermat masalah ini sebelum mulai menuliskan algoritmanya.
   Contoh:
   Arsip 1 mempunyai kunci : 1, 2, 3, 5, 7, 8, 9, 17, 25, 999
   Arsip 2 mempunyai kunci : 2, 3, 15, 17, 81, 90, 999
   Arsip 3 mempunyai kunci : 1, 2, 3, 5, 7, 8, 9, 15, 17, 25, 81, 90, 999

2. Tuliskanlah algoritma umum untuk membuat "*intersection*" rekaman dari dua buah arsip sekuensial yang terurut, yang menghasilkan sebuah arsip terurut dengan kunci unik. Artinya hanya kunci yang sama pada kedua arsip input yang rekamannya ditulis ke arsip hasil. Rekaman yang "dibuang" harus dituliskan ke keluaran dengan perintah output.
   Catatan: Spesifikasikan dengan cermat masalah ini sebelum mulai menuliskan algoritmanya.
   Contoh:
   Arsip 1 mempunyai kunci : 1, 2, 3, 5, 7, 8, 9, 17, 25, 999
   Arsip 2 mempunyai kunci : 2, 3, 15, 17, 81, 90, 999
   Arsip 3 mempunyai kunci : 2, 3, 17, 999

3. Kadang-kadang updating tidak dilakukan dari file, melainkan langsung dari alat masukan (misalnya papan kunci), seperti yang terjadi pada pelayanan-pelayanan umum.
   Diberikan sebuah Master file dengan rekaman <key, val>. Buatlah algoritma untuk:
   - membaca sebuah harga key dan val dari papan kunci;
   - mencari rekaman dengan <key, ---> dalam Master file, dan mengupdate data key tersebut jika ada. Lalu, lakukan penambahan rekaman pada transaction file dengan <key, val>. Jika pencarian tidak ketemu, harus diikuti pesan kesalahan.

4. Tuliskanlah algoritma umum untuk memecah sebuah arsip menjadi dua buah arsip dengan kriteria validitas V terhadap beberapa harga dari rekaman.
   Contoh : Peremajaan arsip
   Rekaman Asli adalah <key:integer, Val1,val2,val3:integer>
   Dipecah menjadi:
   Rekaman Valid adalah <key : integer, Val1, Val2,Val3 : integer >
   dengan Val1, Val2, Val3 memenuhi V(Val1,Val2,Val3)
   Rekaman TidakValid adalah <key : integer, Val1, Val2, Val3 : integer >
   dengan Val1, Val2, Val3 tidak memenuhi V(Val1,Val2,Val3)

5. Apa komentar anda tentang algoritma MERGING2 yang teksnya lebih "singkat" daripada MERGING1?

ANALISIS REKURENS DALAM KONTEKS PROSEDURAL

Sebuah program prosedural, juga dapat bersifat rekursif (lihat topik analisis rekurens pada pemrograman fungsional). Suatu entitas disebut rekursif jika dalam pendefinisian entitas tersebut terkandung entitas tersebut. Bahasan lengkap mengenai rekurens dituliskan pada Diktat "Pemrograman Fungsional". Secara konseptual, semua yang dapat diaplikasi dalam pemrograman fungsional akan dapat pula diimplementasi dalam konteks prosedural.

Dalam notasi Algoritmik, dikenal prosedur rekursif, fungsi rekursif, dan struktur data rekursif. Bagian struktur data rekursif akan dibahas pada diktat "Struktur Data".

Dalam konteks prosedural, rekurens harus dipahami dan dipakai, dengan memandangnya dari dua sudut pandang:
- sudut pandang statik, di mana teks program/algoritma mengandung suatu teks yang sama. Pada bagian deklarasi (kamus) maka akan tampak pada definisi type komposisi yang rekursif (seperti pada list, pohon). Sedangkan pada bagian algoritma, akan tampak sebagai sebuah "call" atau pemakaian suatu fungsi/prosedur terhadap dirinya sendiri.
- sudut pandang dinamik, yaitu mekanisme eksekusi yang terjadi akibat adanya teks yang rekursif, yang akhirnya hanya berdampak kepada "*call*". Call yang bersifat rekursif harus dilakukan dengan penuh pertimbangan, dan bahkan cenderung dihindari karena dapat menyebabkan "*stack overflow*" atau ketidakcukupan memori.

Program utama yang tidak mempunyai parameter, tidak mungkin bersifat rekursif. Sebuah prosedur atau fungsi juga dapat bersifat rekursif. Fungsi rekursif yang dikonstruksi berdasarkan definisi rekursif seperti pada contoh perhitungan faktorial, akan merupakan translasi langsung dari notasi fungsional.

Prosedur rekursif memerlukan cara pendefinisian Initial State dan Final State yang cermat, dan juga cara pemanggilan yang perlu diperhatikan. Passing parameter perlu diperhatikan, dan dalam beberapa kasus harus memakai variabel lokal, untuk mendapatkan hasil yang benar.

Beberapa pedoman dalam merancang prosedur rekursif:
- Prosedur yang hanya mempunyai parameter input yang akan di-*"reccur"* serta parameter output yang merupakan hasil komputasi, sebaiknya diimplementasi sebagai fungsi dengan parameter adalah parameter masukan, dan hasil komputasi fungsi yang dimaksudkan akan diberikan pada parameter output.
- Prosedur yang mempengaruhi "state" atau bekerja terhadap nilai global, harap dirancang dengan hati-hati.
- Prosedur yang mempunyai parameter input/output, dapat dicurigai sebagai implementasi prosedur rekursif yang sebetulnya hanya merupakan pemanfaatan rekurens sebagai mekanisme eksekusi terhadap pengulangan, yang seharusnya dihindari (akan lebih efisien dan jelas konstruksinya sebagai pengulangan).

Contoh dari fungsi dan prosedur rekursif yang dikonstruksi dari definisi rekursif maupun dari mekanisme pengulangan akan diberikan untuk perhitungan faktorial dalam beberapa versi berikut ini:

```
Program ContohRekursif
{ Program ini merupakan contoh implementasi faktorial dalam bentuk }
{ prosedur dan fungsi }
```

```
KAMUS
   function Faktorial (n : integer) → integer
   { Prekondisi : n > 0 }
   { Faktorial(n) menghasilkan 1 jika n = 1;
     menghasilkan n!= n* (n-1)! untuk n>1 }

   procedure Pfaktorial1 (input n : integer, output Hsl : integer)
   { I.S. n > 0 }
   { F.S. Pfaktorial1(n) menghasilkan 1 jika n = 1;
          menghasilkan n! = n*(n-1)! untuk n > 1 }

   procedure Pfaktorial2 (input n : integer, input/output Hsltemp : integer)
   { I.S. n > 0 }
   { F.S. Pfaktorial2(n) menghasilkan 1 jika n = 1;
          menghasilkan n! = n*(n-1)! untuk n > 1, dengan paramater I.O }

   { variabel global program }
   Hsl, HslTemp : integer
```

```
ALGORITMA
   output (Faktorial(5));

   Pfaktorial1(8, Hsl); output (Hsl)

   HslTemp ← 1  { harus diinisialisasi dengan invarian sebab parameter I/O }
   Pfaktorial2(8, HslTemp); output (HslTemp)
```

```
function Faktorial (n : integer) → integer
{ Prekondisi : n > 0 }
{ Faktorial(n) menghasilkan 1 jika n = 1;
  menghasilkan n! = n*(n-1)! untuk n > 1 }
```

```
KAMUS LOKAL
```

```
ALGORITMA
   if (n = 1) then { basis-1 }
      → 1
   else { rekurens }
      → n * Faktorial(n-1)
```

```
procedure Pfaktorial1 (input n : integer, output Hsl : integer)
{ I.S.  n > 0 }
{ F.S. Pfaktorial1(n) menghasilkan 1 jika n = 1;
       menghasilkan n! = n*(n-1)! untuk n > 1 }
{      dengan variabel lokal untuk menyimpan hasil sementara }
```

```
KAMUS LOKAL
   temp : integer
```

```
ALGORITMA
   if (n = 1) then { basis-1 }
      Hsl ← 1
   else { rekurens }
      Pfaktorial1(n-1, temp)
      Hsl ← n * temp
```

Pada solusi sebagai berikut: parameter prosedur adalah sebuah parameter input/output, yang setiap kali akan menyimpan hasil sementara. Setelah perhitungan selesai, maka merupakan hasil akhir perhitungan nilai faktorial.

Karena parameter input/output, maka pada saat pemanggilan harus diinisialisasi dengan tepat, yang akan menyulitkan pengguna program sebab harus mengerti nilai invarian yang tepat agar tidak menghasilkan program yang salah. Ada yang aneh dengan program rekursif semacam ini (dibandingkan dengan solusi sebelumnya di mana Hsl merupakan parameter output sehingga tidak perlu diinisialisasi).

```
procedure Pfaktorial2 (input n : integer, input/output HslTemp : integer)
{ I.S. n > 0 }
{ F.S. Pfaktorial2(n) menghasilkan 1 jika n = 0 atau n = 1;
       menghasilkan n!= n* (n-1)! untuk n>1 }
{ dengan hasil sementara selalu disimpan pada parameter I/O }
{ Nilai hasil harus diinisialisasi dengan 1 saat pemanggilan }
```

```
KAMUS LOKAL
```

```
ALGORITMA
   if (n = 1) then
      { do nothing : kenapa ? }
   else
      Pfaktorial2(n-1, HslTemp)
      HslTemp ← n * HslTemp
```

Call Rekursif sebagai Mekanisme Mengulang

Dalam konteks prosedural, kita dapat mempunyai *loop* sebagai mekanisme untuk mengulang. Salah satu solusi perhitungan faktorial tanpa membuat fungsi rekursif tetapi dengan mengelaborasi solusi menjadi deret kali yang dapat dipakai untuk mengkonstruksi pengulangan: 1 x 2 x 3 x … x N. Tentunya, sebuah prosedur yang setara (dengan *loop*) juga dapat dibuat sebagai gantinya fungsi.

Tampak bahwa dalam solusi ini, setiap kali dilakukan pengulangan, maka nilai perhitungan sementara disimpan dalam suatu variabel lokal.

Kita dapat "mensimulasi" mekanisme eksekusi pengulangan ini dengan melakukan suatu "call"/pemanggilan rekursif, seperti yang akan ditunjukkan dalam contoh-contoh berikut, dengan definisi faktorial(n) untuk n > 0.

Perhatikan bahwa parameter aktual yang diasosiasikan terhadap parameter formal yang bersifat output atau input/output, harus dilakukan dengan memberikan nama parameter dan bukan suatu nilai. Parameter aktual berupa sebuah nilai ekspresi hanya akan valid jika parameter formal bersifat input. Parameter formal fungsi selalu bersifat input.

```
function FaktIter1 (N : integer) → integer
{ Prekondisi : N > 0 }
{ Faktorial (N) menghasilkan 1 jika N = 0 atau N = 1;
  menghasilkan N!= N * (N-1)! untuk  N > 1 dengan loop  }
{ Faktorial adalah perhitungan deret:  1 x 2 x 3 x 4 x … x N }
{ sehingga hasil disimpan dalam Hsl, dan elemen pengulangan adalah I}
```

```
KAMUS LOKAL
   I : integer
   Hsl : integer
```

```
ALGORITMA
   Hsl ← 1       { inisialisasi }
   I ← 1         { invarian : faktorial (1) = 1 }
   while (I ≤ N) then
      Hsl ← Hsl * I
      I ← I + 1 { next-elmt }
   { I > N : sudah dilakukan 1 x 1 x 2 x 3 x … x N }
   → Hsl
```

```
function FaktIter2 (N : integer) → integer
{ Prekondisi : N > 0 }
{ Faktorial (N) menghasilkan 1 jika N = 0 atau N = 1;
  menghasilkan N!= N * (N-1)! untuk n > 1 dengan loop  }
{ Faktorial adalah perhitungan deret : 1 x 2 x 3 x 4 x … x N }
{ Pada versi ini, elemen pengulangan I pada versi FakIter1 dihilangkan
sehingga hasil disimpan dalam Hsl, dan elemen pengulangan adalah N merupakan
versi yang tidak disarankan karena mengubah nilai N dalam body fungsi, yang
karena dalam konsep fungsi semua parameter harus input maka setelah eksekusi
fungsi, nilai N tidak berubah }
```

```
KAMUS LOKAL
   Hsl : integer
```

```
ALGORITMA
   Hsl ← 1;  { inisialisasi }
   { invarian : faktorial (1) = 1 }
   while (N ≥ 1) then
      Hsl ← Hsl * N
      N ← N - 1 { next-elmt }
   { N = 0 : sudah dilakukan 1 x N x (N-1) x (N-2) x … x 3 x 2 x 1 }
   → Hsl
```

```
procedure FakIterRek1 (input N : integer, input/output I : integer,
                       output Hsl : integer)
{ I.S. : N > 0 }
{        Nilai I harus diinisialisasi 1 dan
         Hsl juga diinisialisasi 1 sebelum pemanggilan }
{ F.S. : Faktorial (N) menghasilkan 1 jika N = 0 atau N = 1;
         menghasilkan N!= N*(N-1)! untuk N > 1 dengan loop yang mekanismenya
         dilakukan dengan call rekursif }
{ Versi ini merupakan prosedur, dan tidak mungkin sebagai fungsi,
  karena adanya parameter input/output }
```

KAMUS LOKAL

ALGORITMA

```
   if (I > N) then    { stop condition }
      { do nothing }
   else
      Hsl ← I * Hsl  { badan loop }
      I ← I + 1          { next element }
      FakIterRek1(N, I, Hsl)
```

```
procedure FakIterRek2 (input/output N : integer, output Hsl : integer)
{ I.S. : N > 0 }
{        Hsl harus diinisialisasi  1 sebelum pemanggilan dan harus sesuai
         dengan nilai N. Kesalahan inisialisasi menimbulkan kesalahan
         komputasi }
{ F.S. : Faktorial (N) menghasilkan 1 jika  N = 1;
         menghasilkan N!= N * (N-1)! untuk N > 1 dengan loop yang
         mekanismenya  dilakukan dengan call rekursif }
{        1 * N * N(-1) * (N-2) * … x 3 x 2 x 1 }
{ Versi ini merupakan prosedur, dan tidak mungkin sebagai fungsi,
  karena adanya parameter input/output }
```

KAMUS LOKAL

ALGORITMA

```
   if (N = 1) then    { basis = stop condition }
      { do nothing }
   else               { N > 1 }
      Hsl ← N * Hsl  { badan loop }
      N ← N - 1      { next element }
      FakIterRek2(N, Hsl)
```

Perhatikan bahwa realisasi prosedur FakIterRek1 dan FakIterRek2 di atas sangat tidak natural, karena merupakan *loop* yang dieksekusi dengan mekanisme *call* rekursif. Pengguna prosedur juga harus mengetahui nilai inisialisasi parameter input/output agar prosedur dapat dieksekusi dengan benar. Kesalahan menginisialisasi nilai parameter aktual akan mengakibatkan kesalahan hasil komputasi. Ini juga bukan merupakan prosedur yang baik, karena dapat mengakibatkan kesalahan.

Dalam pemrograman prosedural, pengulangan lebih natural jika diimplementasi menjadi *loop*. Sebuah *call* rekursif atau pemanggilan apapun, menjadi tidak natural jika pengguna harus memberikan nilai inisialisasi yang tepat untuk suatu parameter input/output.

Realisasi prosedur/fungsi rekursif yang baik selalu didasari oleh definisi rekursif. Dalam pemrograman yang diberikan untuk Mesin Gambar (lihat bab terkait) dan pemrograman yang lebih lanjut, Anda akan mendapatkan pola-pola program rekursif yang memang didasari fungsi rekursif

Studi Kasus Pemrosesan Tabel secara Rekursif

Pada bagian berikut ini, akan diberikan skema pemrosesan elemen secara rekursif, dengan melakukan rekurens terhadap rentang nilai (interval) indeks elemen yang diproses. Dalam kasus ini, array didefinisikan sebagai variabel global, dan tidak dipassing sebagai parameter karena array merupakan data global yang mencerminkan state dari sistem. Prosedur Print yang direalisasi sebagai contoh tidak direaliasi sebagai fungsi, sebab mencetak adalah aksional (bukan fungsional). Ini merupakan contoh situasi dimana prosedur rekursif dibutuhkan, dan sangat dekat dengan bentuk pengulangan. Beberapa algoritma lanjut yang akan dipelajari pada perkuliahan selanjutnya akan bekerja pada tabel secara rekursif (misalnya quick sort).

```
KAMUS GLOBAL
    T : array [1..IdxMax] of integer
    Neff : integer   { tabel T terisi dari 1 s.d. Neff }
    { T dan Neff adalah variabel global yang merupakan state dari sistem }
    { dan hanya ada 1 copy tabel }
```

```
procedure PrintTab (input IAwal, IAkhir : integer)
{ Prosedur ini mencetak nilai elemen tabel global T untuk indeks}
{ IAwal s.d. IAkhir }
{ I.S. IAwal ≤ IAkhir dan berada dalam range [1..Neff] }
{      Tabel T adalah tabel global yang sudah terdefinisi }
```

```
KAMUS LOKAL
```

```
ALGORITMA
    if (IAwal > IAkhir) then     { basis kosong = stop condition }
        { do nothing }
    else                         { IAwal ≤ IAkhir, rekurens }
        output (T[IAwal])        { current element }
        PrintTab(IAwal+1,IAkhir) { IAwal dapat di-passing sebagai ekspresi }
                                 { karena merupakan parameter input }
```

Latihan Soal

1. Pelajari kembali mekanisme passing parameter dan nilai variabel lokal dalam bahasa pemrograman yang dipakai sebagai latihan.
2. Pelajarilah kembali semua solusi rekursif untuk persoalan perhitungan nilai faktorial (N) di atas, dan modifikasilah untuk dapat memenuhi spesifikasi, bahwa nilai faktorial(0) adalah 1.
3. Buatlah prosedur rekursif yang akan menghitung fibonacci (carilah definisi rekursif dari fungsi Fibonacci).
4. Pelajarilah prosedur mem-*print* tabel yang diberikan pada studi kasus:
   a. Apa akibatnya jika parameter prosedur bersifat input/output ?
   b. Bagaimana jika dikehendaki untuk mengaplikasi basis-1 terhadap tabel tersebut?
   c. Bagaimana jika tabel bukan merupakan variabel global?
5. Berdasarkan contoh memprint tabel sehingga:
   a. Mem-*print* secara mundur (mulai dari elemen IAkhir s.d. IAwal.
   b. Memeriksa apakah ada elemen tabel yang bernilai X.
   c. Andaikata elemen tabel yang terisi adalah IdxMin s.d. Neff, maka tuliskan prosedur untuk:

i. Menambahkan sebuah elemen menjadi elemen setelah Neff.
ii. Menambahkan sebuah elemen di awal tabel, sehingga menggeser semua elemen tabel dan Neff bertambah dengan satu.

6. Buatlah pola program untuk mengolah nilai matriks (tabel dua dimensi) secara rekursif.
7. Mungkinkah untuk mengolah file sekuensial secara rekursif?

TERJEMAHAN NOTASI ALGORITMIK KE BAHASA PASCAL

Notasi algoritmik dipakai untuk mengembangkan solusi, dan memang dirancang tanpa pengeksekusi. Salah satu bahasa pengeksekusi yang sangat dekat dengan notasi algoritmik adalah bahasa Pascal *standard*.

Pada bagian ini diberikan terjemahan notasi algoritmik ke dalam bahasa Pascal. Mahasiswa disarankan untuk melakukan translasi semua teks algoritmik menjadi program Pascal, dan berlatih mengeksekusinya dengan menyediakan data test yang sesuai.

ALGORITMA	PASCAL
1. ASSIGNMENT: <nama> ← <harga>	 <nama> := <harga>;

2. KONDISIONAL:
 	
if <kondisi> then
 	
Aksi-A
	
	if (kondisi) **then**
	begin
	Aksi-A
	end;

if <kondisi> then
 	
Aksi-A
else
 	
Aksi-B
	if (kondisi) **then**
	begin
	Aksi-A;
	end else
	begin
	Aksi-B;
	end;

depend on <nama-var>
<kondisi-1> : Aksi-1
<kondisi-2> : Aksi-2
<kondisi-3> : Aksi-3
.
.
.
<kondisi-n> : Aksi-n
	if (kondisi-1) **then**
	begin
	Aksi-1;
	end else if (kondisi-2) **then**
	begin
	Aksi-2;
	end else if (kondisi-3) **then**
	begin
	Aksi-3;
	end else if (kondisi-4) **then**
	begin
	.
	.
	.
	end else if (kondisi-n) **then**
	begin
	Aksi-n;
	end;

Catatan:
Untuk depend-on, nama-var adalah type yang dapat dienumerasi, maka pada Pascal dapat dipakai statement case:

```
case (nama-var) of
      konstan-1 : Aksi-1;
      konstan-2 : Aksi-2;
      konstan-3 : Aksi-3;
      else Aksi-4
end;
```

ALGORITMA	PASCAL

3. PENGULANGAN:
 	
while <kondisi-ULANG> do
 	
Aksi
	
	while (kondisi-ULANG) **do**
	begin
	Aksi
	end;

repeat Aksi until <kondisi-STOP>	**repeat** Aksi **until** (kondisi-STOP);

iterate
 	
Aksi-A
 	
stop <kondisi-STOP>
 	
Aksi-B
	(* deklarasi stop : boolean *)
	stop := false;
	repeat
	Aksi-A;
	if (kondisi-STOP) **then**
	stop := true
	else
	Aksi-B;
	until (stop);

i traversal [Awal..Akhir]
 	
Aksi
	(* Jika Awal <= Akhir *)
	for (i := Awal **to** Akhir) **do**
	begin
	Aksi;
	end;
	
	(* Jika Awal >= Akhir *)
	for (i := Awal **downto** Akhir) **do**
	begin
	Aksi;
	end;

4. INPUT/OUTPUT: input <nama>	 **read**(nama); **readln**(nama);
output <nama>	**write**(nama); **writeln**(nama);

Catatan:
Perhatikan hal-hal yang harus diperhatikan (penulisan pesan) dalam input/output, agar pengguna program tidak perlu membaca *source code* untuk dapat memberikan input yang benar.

PROTOTYPE PROSEDUR DAN FUNGSI

Pendefinisian/Spesifikasi Fungsi

(Notasi Algoritmik diberikan sebagai komentar, langsung dituliskan dalam bahasa Pascal)

```
(* function NAMAF (<list parameter input>) → <type hasil> *)

function NAMAF (<list parameter input>) : <type hasil>
(* Spesifikasi fungsi: diberikan ... menghasilkan ... *)
(* dalam bahasa Pascal standard, type hasil harus type primitif, *)
(* atau untuk array diberi nama *)
```

```
(* KAMUS LOKAL: boleh mengandung VAR, TYPE, CONST *)

VAR
(* semua NAMA yang dipakai dalam algoritma/realisasi fungsi *)
```

```
(* ALGORITMA *)
begin
    (* Deretan instruksi algoritmik :
       pemberian harga, input, output, analisis kasus, pengulangan *)

    (* Pengiriman harga di akhir fungsi, harus sesuai dengan type hasil *)
    NamaF := <ekspresi hasil>;
end;
```

Pemanggilan Fungsi

(Langsung dituliskan dalam bahasa Pascal)

```
(* Program POKOKPERSOALAN *)
Program POKOKPERSOALAN;
(* Spesifikasi : Input, Proses, Output *)
```

```
(* KAMUS *)

(* Semua NAMA yang dipakai dalam program *)
(* TYPE  *)
(* CONST *)
(* VAR   *)

(*  Spesifikasi dan body fungsi langsung dituliskan di bagian ini *)
```

```
(* ALGORITMA *)
begin
    (* Deretan instruksi pemberian harga, input, output, analisis kasus,
       pengulangan yg memakai fungsi *)

    (* Harga yang dihasilkan fungsi juga dapat dipakai dalam ekspresi *)

    nama := NAMAF(<list parameter aktual>);

    write(NAMAF(<list parameter aktual>));

    (* Parameter aktual dapat berupa nilai, konstanta, ekspresi *)
    (* Harga yang dihasilkan fungsi juga dapat dipakai dalam ekspresi *)
end.
```

Pendefinisian/Spesifikasi Prosedur

```
(* procedure NAMAPROSEDUR (input <list nama parameter formal : type>,
                           input/output <list nama parameter formal : type>,
                           output <list nama parameter formal : type>)
*)

procedure NAMAPROSEDUR ( <list nama parameter formal input: type>;
                         var <list nama parameter formal input/output atau
                             output> : type>);

(* Spesifikasi, Initial State, Final State *)
```

```
(* KAMUS LOKAL: boleh mengandung VAR, TYPE, CONST *)
(* Semua NAMA yang dipakai dalam BADAN PROSEDUR *)
```

```
(* ALGORITMA *)
begin
   (* BADAN PROSEDUR *)
   (* Deretan instruksi pemberian harga, input, output, analisis kasus,
      pengulangan atau prosedur *)
end;
```

Pemanggilan Prosedur

```
(* Program POKOKPERSOALAN *)
Program POKOKPERSOALAN;
(* Spesifikasi, Input, Proses, Output *)
```

```
(* KAMUS *)

(* Semua NAMA yang dipakai dalam program }
(* TYPE  *)
(* CONST *)
(* VAR   *)

(*  Spesifikasi dan body prosedur langsung dituliskan di bagian ini *)

(* procedure NAMAPROSEDUR (input/output <list nama parameter formal>) *)
procedure NAMAPROSEDUR (<list-nama parameter formal input : type;
                         var <list-nama parameter formal input/output atau
                             output> : type>);
(* Spesifikasi : Initial State, Final State *)
(* Boleh mengandung kamus lokal VAR, TYPE, CONST *)
begin

end;
```

```
(* ALGORITMA PROGRAM UTAMA *)
begin
   (* Deretan instruksi assignment/pemberian harga, input, output, analisis
      kasus, pengulangan *)

   NAMAPROSEDUR(<list parameter aktual>);
end.
```