UDAYANA UNIVERSITY PRESS

KONSEP PEMROGRAMAN KOMPUTER

BERBASIS TEKS DAN GRAFIS

IR. MADE SUDARMA, M.A.SC.

KONSEP PEMROGRAMAN KOMPUTER

Undang-Undang Republik Indonesia Nomor 19 Tahun 2002 Tentang Hak Cipta

Lingkup Hak Cipta
Pasal 2

1. Hak Cipta merupakan hak eksklusif bagi Pencipta atau Pemegang Hak Cipta untuk mengumumkan atau memperbanyak Ciptaannya, yang timbul secara otomatis setelah suatu ciptaan dilahirkan tanpa mengurangi pembatasan menurut peraturan perundang-undangan yang berlaku.

Ketentuan Pidana
Pasal 72

1. Barang siapa dengan sengaja melanggar dan tanpa hak melakukan perbuatan sebagaimana dimaksud dalam Pasal 2 Ayat (1) atau Pasal 49 Ayat (1) dan Ayat (2) dipidana dengan penjara masing-masing paling singkat 1 (satu) bulan dan/atau denda paling sedikit Rp 1.000.000,00 (satu juta rupiah), atau pidana penjara paling lama 7 (tujuh) tahun dan/atau denda paling banyak Rp 5.000.000,00 (lima juta rupiah).
2. Barang siapa dengan sengaja menyiarkan, memamerkan, mengedarkan atau menjual kepada umum suatu ciptaan atau barang hasil pelanggaran hak cipta atau hak terbit sebagai dimaksud pada Ayat (1) dipidana dengan pidana penjara paling lama 5 (lima) tahun dan/atau denda paling banyak Rp. 500.000.000,00 (lima ratus juta rupiah).

KONSEP PEMROGRAMAN KOMPUTER

BERBASIS TEKS DAN GRAFIS

Ir. Made Sudarma, M.A.SC.

UDAYANA UNIVERSITY PRESS
2012

KONSEP PEMROGRAMAN KOMPUTER

BERBASIS TEKS DAN GRAFIS

Penulis:
Ir. Made Sudarma, M.A.SC.

Penyunting:
Jiwa Atmaja

Cover & Ilustrasi:
Repro

Design & Lay Out:
Putu Mertadana

Diterbitkan oleh:
Udayana University Press
Kampus Universitas Udayana Denpasar
Jl. P.B. Sudirman, Denpasar - Bali, Telp. 0361 9173067, 255128
Fax. 0361 255128
Email: unudpress@yahoo.com http://penerbit.unud.ac.id

Cetakan Pertama:
2012, x + 253 hlm, 14 x 21 cm

ISBN: 978-602-9042-52-8

PRAKATA

Buku ini semula berupa naskah yang didisusun secara khusus untuk keperluan pengajaran kuliah Algoritma dan Pemrograman di lingkungan Teknik Elektro Bidang Keahlian Sistem Komputer dan Informatika, Fakultas Teknik Universitas Udayana. Jadi, buku ini merupakan revisi dari diktat yang pernah disusun untuk perkuliahan Algoritma dan Pemrograman, yang sangat erat hubungannya dengan matakuliah terkait, yaitu Pemrograman Fungsional, Struktur Data, dan Pemrograman Berorientasi Objek.

Buku ini disusun dengan tujuan untuk membekali mahasiswa 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 buku ini, 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.

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

Denpasar, 16 Januari 2012
Penulis

DAFTAR ISI

BAB I
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 apa pun, walaupun implementasi dan cara konstruksinya akan sangat tergantung kepada paradigma dan bahasa pemrogramannya.

1.1 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. Bagaimanapun fanatisme terhadap sebuah paradigma, mempersempit wawasan dan kadang berbahaya.

Dalam pemrograman pun ada beberapa paradigma, masing-masing mempunyai prioritas 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 mungkin tidak cocok. Mengharuskan seseorang memecahkan persoalan hanya dengan melalui sebuah paradigma, berarti membatasi strateginya dalam pemrograman. Satu paradigma tidak akan cocok untuk semua kelas persoalan.

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

Paradigma ini didasari oleh konsep mesin Von Newmann (*stored program concept*) : sekelompok tempat

penyimpanan (*memory)*, yang dibedakan menjadi memori *instruksi* dan *memory 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 dari pada 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 logik*

Paradigma ini didasari oleh pendefinisian relasi antar individu yang dinyatakan sebagai "*predikat*". Sebuah program logik 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, metode). 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 objek yang berada dalam kelas yang sama. Kelas-kelas mempunyai hierarki, 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 berinterasi. Definisi kelas memungkinkan adanya penurunan kelas dengan objek pada saat *run time* 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, mengkonstruksi program dari objek dan kelas adalah berbeda dengan mengkonstruksi 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.

1.2 Bahasa Pemrograman

Ada banyak sekali bahasa pemrograman, mulai dari bahasa tingkat rendah (bahasa mesin dalam biner), bahasa *asembler (*dalam kode menemonik), bahasa tingkat tinggi, sampai bahasa generasi ke empat (4GL).

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

1. Prosedural : Algol, Pascal, Fortran, Basic, Cobol, C ...
2. Fungsional : LOGO, APL, LISP
3. Deklaratif/Logik : Prolog
4. Object oriented murni: Smalltalk, Eifel, Jaca, C++..
5. Konkuren : OCCAM, Ada, Java
6. Relasional: SQL pada basis data relasional

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

1.3 Belajar Pemrograman Tidak Sama dengan Belajar Bahasa Pemrograman

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

Pada hakikatnya, penggunaan komputer untuk memecahkan persoalan adalah untuk tidak mengulang-ulang kembali hal yang sama. Strategi pengenalan masalah melalui dekomposisi, pemakaian kembali modul yang ada, sintesis, 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, tetapi relatif lebih sedikit dibandingkan dengan bongkar pasang program yang sudah ada.

Belajar memprogram lebih bersifat pemahaman persoalan, analisis, sintesis. Belajar bahasa pemrograman adalah belajar memakai suatu bahasa, aturan sintaks (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 memprogram lebih bersifat keterampilan dari pada analisis dan sintesis.

Belajar memprogram dan belajar bahasa pemrograman mempunyai tingkatan dan kesulitan yang

berbeda-beda. Mahasiswa seringkali dihadapkan pada kedua kesulitan itu sekaligus. Pemecahan persoalan dengan paradigma yang sama akan menghasilkan solusi yang "sejenis". Beberapa bahasa dapat termasuk dalam sebuah paradigma sama, pemecahan persoalan dalam satu paradigma dapat diterjemahkan ke dalam bahasa-bahasa yang berbeda. Untuk itulah diperlukan adanya suatu perjanjian, notasi yang disepakati supaya masalah itu dapat dengan mudah diterjemahkan ke dalam salah satu bahasa yang masih ada dalam lingkup paradigma yang sama.

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 praktik 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 praktiknya, 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.

Metode 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.

1.4 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 dipoles 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 standar. Sikap mahasiswa yang mau disiplin secara mandiri sangat diharapkan dalam mencapai tujuan kuliah ini.

1.5 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 atas komponen-komponen "kecil". Misalnya, sebuah peralatan elektronik yang canggih, di dalamnya terdiri atas komponen elektronik "standar" 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. 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*, dsb)

Pada 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.

1.6 Pemrogram Individu dan Pemrogram dalam Tim

Jaman dulu, tidak jarang ditemui *programmer* "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].

Jaman 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 standar 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. Aspek ini perlu diperhatikan dan dengan disepakati bersama.

Aspek bekerja dalam tim dan mengikuti standar ini akan dicakup dalam perkuliahan dasar pemrogramam. Jadi, salah satu tujuan pengajaran adalah membentuk pemrogram atau perancang program yang harus mampu bekerja sama dengan orang lain. Kuliah Algoritma Pemrograman lebih diarahkan kepada *programming*, daripada *coding*.

Beberapa kalangan berpendapat "bahwa memprogram adalah menghasilkan program". Ini dapat dianalogikan dengan melihat seorang penyanyi, maka 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, dsb. 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 yang perlu dilakukan mahasiswa yang sedang belajar memprogram 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 petunjuk 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 penterjemahan 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 logik 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 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 memprogram. 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.

1.7 Tujuan Pemrograman Prosedural

Tujuan utama dari pembelajaran 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 apa pun, 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 apa pun.

2. Menulis algoritma dari suatu masalah dengan menggunakan metodologi dan skema standar 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 dibahas 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.

BAB II
PENGENALAN PEMROGRAMAN KOMPUTER

Bagian ini akan membahas dasar – dasar komponen dari komputer meliputi *hardware* (perangkat keras) dan *software* (perangkat lunak). Kami juga akan menyertakan gambaran global tentang bahasa pemrograman dan sirkulasi pemrograman. Akan dibahas pula pada akhir pembahasan ini mengenai sistem dan konversi numerik.

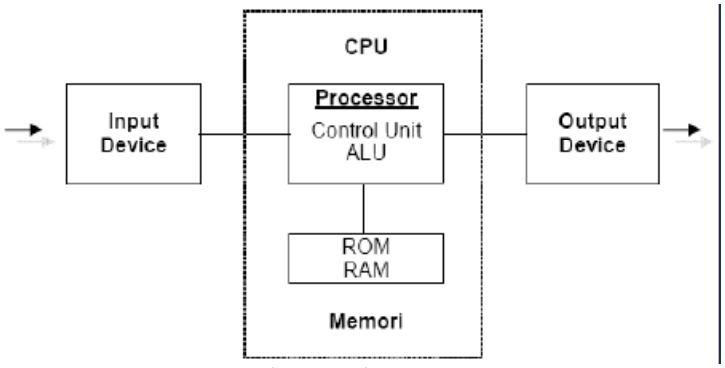
Pada akhir pembahasan, diharapkan pembaca dapat:

- Mengindentifikasi perbedaan komponen pada komputer
- Mengetahui tentang bahasa pemrograman komputer dan kategorinya
- Mengetahui alur kerja pembuatan program dan mengaplikasikannya pada pemecahan masalah
- Mengetahui tentang sistem numerik dan metode konversinya.

2.2 Pendahuluan

Kata komputer berasal dari bahasa Latin yaitu *Computare* yang artinya "menghitung". Dalam bahasa Inggris disebut *to compute.* Secara definisi komputer diterjemahkan sebagai sekumpulan alat elektronik yang

saling bekerja sama, dapat menerima data (input), mengolah data (proses) dan memberikan informasi (output) serta terkoordinasi dibawah kontrol program yang tersimpan di memorinya. Jadi cara kerja komputer dapat kita gambarkan sebagai berikut :



Skema I/O komputer

Komputer memiliki dua komponen utama. Yang pertama adalah *hardware* (perangkat keras) yang tersusun atas komponen elektronik dan mekanik.

Komponen utama yang lain yaitu *software* (perangkat lunak). Komponen ini terdiri atas data dan aplikasi – aplikasi komputer.

2.3 Komponen Dasar Komputer

2.3.1 *Hardware*

2.3.1.1 Central Processing Unit (CPU)

Processor, merupakan bagian dari perangkat keras komputer yang melakukan pemprosesan aritmatika, dan logika, serta pengendalian operasi komputer secara keseluruhan. Prosesor terdiri atas dua bagian utama, yaitu

ALU (*Arithmetic Logic Unit*) dan *Control Unit*. Kecepatan kerja prosesor biasanya ditentukan oleh kecepatan *clock* dari *Control Unit*-nya. Contoh, : jika prosesor memiliki frekuensi *clock* 350 MHz, berarti kecepatan pemprosesan satu instruksinya = T = 1/f = 1/(350 x 106 Hz), = 0,286 x 10-8 detik.

2.3.1.2 ***Memory***

Memory adalah media penyimpan data pada komputer. Memory, berdasarkan fungsinya dibagi menjadi dua, yaitu :

1a. ***Primary Memory***

Dipergunakan untuk menyimpan data dan instruksi dari program yang sedang dijalankan. Biasa juga disebut sebagai RAM. Karakteristik dari memori primer adalah :

- Volatil (informasi ada selama komputer bekerja. Ketika komputer dipadamkan, informasi yang disimpannya juga hilang).
- Berkecepatan tinggi.
- Akses random (acak).

2b. ***Secondary Memory***

Dipergunakan untuk menyimpan data atau program biner secara permanen. Karakteristik dari memori sekunder adalah

- Non volatil atau persisten
- Kecepatan relatif rendah (dibandingkan memori primer)
- Akses random atau sekuensial

3 Contoh memori sekunder : *floppy*, *harddisk*, *CD ROM*, *magnetic tape, optical disk*, dll. Dari seluruh contoh

tersebut, yang memiliki mekanisme akses sekuensial adalah *magnetic tape.*

Tabel 1: Perbandingan antara memori utama dan memori sekunder

Memori Utama (RAM)	***Memori Sekunder (ROM)***	***Kategori***
Cepat	Lambat	Kecepatan
Mahal	Murah	Harga
Kecil	Besar	Kapasitas
Ya	Tidak	Volatile

2.3.1.3 *Input* dan *Output Device*

Input-Output Device, merupakan bagian yang berfungsi sebagai penghubung antara komputer dengan lingkungan di luarnya. Dapat dibagi menjadi dua kelompok, yaitu

1a. *Input Device* (Piranti Masukan)

Berfungsi sebagai media komputer untuk menerima masukan dari luar. Beberapa contoh piranti masukan :

- *Keyboard*
- *Mouse*
- *Touch screen*
- *Scanner*
- *Camera*

2b. *Output Device* (Piranti Keluaran)

Berfungsi sebagai media komputer untuk memberikan keluaran. Beberapa contoh piranti keluaran :

- *Monitor*
- *Printer*

- *Speaker*
- *Plotter*

2.3.2 *Software*

Merupakan program-program komputer yang berguna untuk menjalankan suatu pekerjaan sesuai dengan yang dikehendaki. Program tersebut ditulis dengan bahasa khusus yang dimengerti oleh komputer. Program dapat dianalogikan sebagai instruksi yang akan dijalankan oleh prosessor. *Software* terdiri dari beberapa jenis, yaitu :

1. **Sistem Operasi,** seperti DOS, Unix, Novell, OS/2, Windows.
   Adalah *software* yang berfungsi untuk mengaktifkan seluruh perangkat yang terpasang pada komputer sehingga masing-masingnya dapat saling berkomunikasi. Tanpa ada sistem operasi maka komputer tidak dapat difungsikan sama sekali.
2. **Program *Utility*,** seperti ***Norton Utility*, *Scandisk, PC Tools.***
   Program *utility* berfungsi untuk membantu atau mengisi kekurangan/kelemahan dari sistem operasi, misalnya PC Tools dapat melakukan perintah format sebagaimana DOS, tetapi PC Tools mampu memberikan keterang dan animasi yang bagus dalam proses pemformatan. File yang telah dihapus oleh DOS tidak dapat dikembalikan lagi, tetapi dengan program bantu hal ini dapat dilakukan.
3. **Program Aplikasi,** seperti ***GL, MYOB, Payroll.***
   Merupakan program yang khusus melakukan suatu pekerjaan tertentu, seperti program gaji pada suatu perusahaan. Program ini hanya digunakan oleh bagian keuangan saja tidak dapat digunakan oleh departemen yang lain. Umumnya program aplikasi

ini dibuat oleh seorang *programmer* komputer sesuai dengan permintaan/kebutuhan seseorang/lembaga/ perusahaan guna keperluan interennya.

4. **Program Paket**

   Merupakan program yang dikembangkan untuk kebutuhan umum, seperti :

   - Pengolah kata /editor naskah : *Wordstar, MS Word, Word Perfect, AmiPro*
   - Pengolah angka / lembar kerja : *Lotus123, MS Excell, QuattroPro*, dll
   - Presentasi : *MS Power Point*
   - Desain grafis : *Corel Draw, PhotoShop*

5. **Compiler**

   Komputer hanya memahami satu bahasa, yaitu bahasa mesin. Bahasa mesin adalah terdiri dari nilai 0 dan 1. Sangatlah tidak praktis dan efisien bagi manusia untuk membuat program yang terdiri dari nilai 0 dan 1, maka dicarilah suatu cara untuk menterjemahkan sebuah bahasa yang dipahami oleh manusia menjadi bahasa mesin. Dengan tujuan inilah, diciptakan *compiler.*

2.4 Sekilas Bahasa Pemrograman

2.4.1 Apa yang Disebut Bahasa Pemrograman?

Bahasa pemrograman adalah teknik komunikasi standar untuk mengekspresikan instruksi kepada komputer. Layaknya bahasa manusia, setiap bahasa memiliki tata tulis dan aturan tertentu. Bahasa pemrograman memfasilitasi seorang *programmer* untuk secara spesifik apa yang akan dilakukan oleh komputer selanjutnya, bagaimana data tersebut disimpan dan dikirim, dan apa yang akan dilakukan apabila terjadi kondisi yang variatif. Bahasa

pemrograman dapat diklasifikasikan menjadi tingkat rendah, menengah, dan tingkat tinggi. Pergeseran tingkat dari rendah menuju tinggi menunjukkan kedekatan terhadap ”bahasa manusia”.

2.4.2 Kategori Bahasa Pemrograman

1. **Bahasa Pemrograman Tingkat Tinggi**
   Merupakan bahasa tingkat tinggi yang mempunyai ciri-ciri mudah dimengerti karena kedekatannya terhadap bahasa sehari – hari. Sebuah pernyataan program diterjemahkan kepada sebuah atau beberapa mesin dengan menggunakan ***compiler***. Sebagai contoh adalah : JAVA, C++, .NET
2. **Bahasa Pemrograman Tingkat Rendah**
   Bahasa pemrograman generasi pertama. Bahasa jenis ini sangat sulit dimengerti karena instruksinya menggunakan bahasa mesin. Disebut juga dengan bahasa *assembly* merupakan bahasa dengan pemetaan satu – persatu terhadap instruksi komputer. Setiap intruksi *assembly* diterjemahkan dengan menggunakan ***assembler.***
3. **Bahasa Pemrograman Tingkat Menengah**
   Di sini, penggunaan instruksi telah mendekati bahasa sehari – hari, walaupun masih cukup sulit untuk dimengerti karena menggunakan singkatan – singkatan seperti STO yang berarti simpan (*STORE*) dan *MOV* yang artinya pindah (*MOVE*). Yang tergolong dalam bahasa ini adalah *Fortran*.

2.5 Alur Pembuatan Program

Seorang *programmer* tidak melakukan pembuatan dan pengkodean program secara begitu saja, tetapi mengikuti perencanaan dan metodologi yang terstruktur yang memisahkan proses suatu aplikasi menjadi beberapa bagian. Berikut ini langkah – langkah sistematis dasar dalam menyelesaikan permasalahan pemrograman :

- Mendefiniskan masalah.
- Menganalisis dan membuat rumusan pemecahan masalah.
- Desain Algoritma dan Representasi.
- Pengkodean, Uji Coba dan pembuatan dokumentasi.

Untuk memahami langkah dasar dalam pemecahan masalah dalam sebuah komputer mari kita mendefinisikan sebuah permasalahan yang akan diselesaikan langkah demi langkah sebagaimana metodologi pemecahan masalah yang akan dibahas selanjutnya. Masalah yang akan kita selesaikan akan didefinisikan pada bagian selanjutnya.

2.5.1 Definisi Permasalahan

Seorang *programmer* umumnya mendapatkan tugas berdasarkan sebuah permasalahan. Sebelum sebuah program dapat terdesain dengan baik untuk menyelesaikan beberapa permasalahan, masalah – masalah yang terjadi harus dapat diketahui dan terdefinisi dengan baik untuk mendapatkan detail persyaratan *input* dan *output*. Sebuah pendefinisan yang jelas adalah sebagian dari penyelesaian masalah. Pemrograman komputer mempersyaratkan untuk mendefiniskan program terlebih dahulu sebelum membuat suatu penyelesaian masalah. Mari kita definisikan sebuah contoh permasalahan:

"Buatlah sebuah program yang akan menampilkan berapa kali sebuah nama tampil pada sebuah daftar"

2.5.2 Analisis Permasalahan

Setelah sebuah permasalahan terdefinisi secara memadai, langkah paling ringkas dan efisien dalam penyelesaian harus dirumuskan. Umumnya, langkah berikutnya meliputi memecahkan masalah tersebut menjadi beberapa bagian kecil dan ringkas.

Contoh masalah : Menampilkan jumlah kemunculan sebuah nama pada daftar

Input Terhadap Program : Daftar Nama, Nama yang akan dicari

***Output* Dari Program :** Jumlah kemunculan nama yang dicari

2.5.3 Desain Algoritma dan Representasi

Setelah kita mengetahui dengan baik dan jelas mengenai permasalahan yang ingin diselesaikan, langkah selanjutnya adalah membuat rumusan algoritma untuk menyelesaikan permasalahan. Dalam pemrograman komputer penyelesaian masalah didefinisikan dalam langkah demi langkah.

Algoritma adalah urutan langkah – langkah logis penyelesaian masalah yang disusun secara sistematis dan logis. Logis merupakan kunci dari sebuah algoritma. Langkah – langkah dalam algoritma harus logis dan bernilai benar atau salah. Algoritma dapat diekpresikan dalam bahasa manusia, menggunakan presentasi grafik melalui sebuah *Flow Chart* (diagram alir) ataupun melalui *Pseudo Code* yang menjembatani antara bahasa manusia dengan bahasa pemrograman.

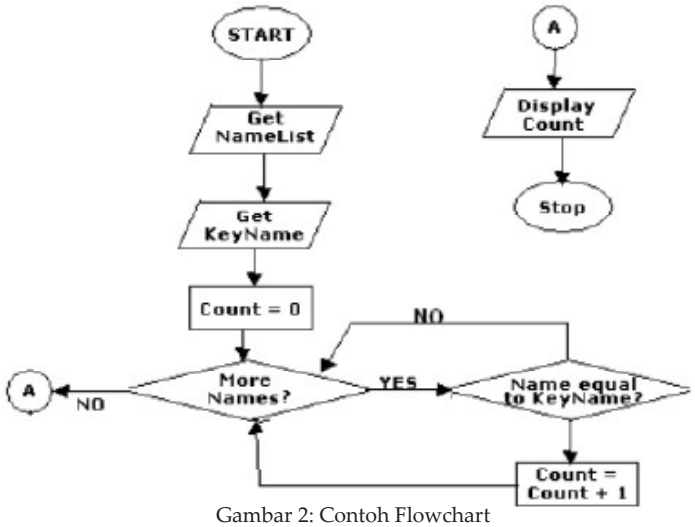
Berdasarkan permasalahan yang terjadi pada bagian

sebelumnya, bagaimanakah kita dapat memberikan solusi penyelesaian secara umum dalam sebuah alur yang dapat dengan mudah dimengerti?

Mengekspresikan cara penyelesaian melalui bahasa manusia :

1. Tentukan daftar nama.
2. Tentukan nama yang akan dicari, anggaplah ini merupakan sebuah kata kunci.
3. Bandingkan kata kunci terhadap setiap nama yang terdapat pada daftar.
4. Jika kata kunci tersebut sama dengan nama yang terdapat pada daftar, tambahkan nilai 1 pada hasil perhitungan.
5. Jika seluruh nama telah dibandingkan, tampilkan hasil perhitungan (*output*).

Mengekspresikan cara penyelesaian melalui FlowChart :



Gambar 2: Contoh Flowchart

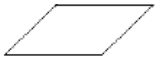
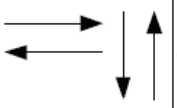
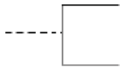
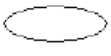
Mengekspresikan solusi melalui *Pseudocode* :
listNama = Daftar Nama
keyNama = Nama yang dicari
hitung = 0
Untuk setiap nama pada Daftar Nama lakukan :
Jika nama == keyNama
Hitung = Hitung + 1
Tampilkan Hitung

2.5.3.1 Simbol *Flowchart* dan Artinya

Flowchart adalah representasi grafis dari langkah – langkah yang harus diikuti dalam menyelesaikan suatu permasalahan yang terdiri atas sekumpulan simbol, di mana masing – masing simbol merepresentasikan kegiatan tertentu. *Flowchart* diawali dengan penerimaan input dan diakhiri dengan penampilan *output*. Sebuah *flowchart* pada umumnya tidak menampilkan instruksi bahasa pemrograman, namun menetapkan konsep solusi dalam bahasa manusia atau pun notasi matematis.

Berikut ini akan dibahas tentang simbol – simbol yang digunakan dalam menyusun *flowchart*, kegiatan yang diwakili serta aturan yang diterapkan dalam penggunaan simbol tersebut :

Tabel 2: Simbol dari *Flowchart*

Simbol	Nama	Pengertian
	**Simbol Proses**	Simbol ini digunakan untuk melambangkan kegiatan pemrosesan input. Dalam simbol ini, kita dapat menuliskan operasi-operasi yang dikenakan pada input, maupun operasi lainnya. Sama seperti aturan pada simbol input, penulisan dapat dilakukan secara satu per satu maupun secara keseluruhan.
	**Simbol Input – Output (IO)**	Merepresentasikan fungsi I/O yang membuat sebuah data dapat diproses (input) atau ditampilkan (output) setelah mengalami eksekusi informasi
	**Simbol Garis Alir**	Simbol ini digunakan untuk menghubungkan setiap langkah dalam flowchart dan menunjukkan kemana arah aliran diagram. Anak panah ini harus mempunyai arah dari kiri ke kanan atau dari atas ke bawah. Anak panah ini juga dapat diberi label, khususnya jika keluar dari symbol percabangan.
	**Simbol Anotasi**	Merepresentasikan informasi deskriptif tambahan, komentar atau catatan penjelasan. Dalam simbol ini, kita dapat menuliskan komentar apapun dan sebanyak apapun, hal ini berguna untuk memperjelas langkah-langkah dalam flowchart. Garis vertical dan garis terputus – putus dapat ditempatkan pada sisi kanan maupun kiri.
	**Simbol Percabangan**	Simbol ini digunakan untuk melambangkan percabangan, yaitu pemeriksaan terhadap suatu kondisi. Dalam simbol ini, kita menuliskan keadaan yang harus dipenuhi. Hasil dari pemeriksaan dalam simbol ini adalah YES atau NO. Jika pemeriksaan menghasilkan keadaan benar, maka jalur yang harus dipilih adalah jalur yang berlabel Yes, sedangkan jika pemeriksaan menghasilkan keadaan salah, maka jalur yang harus dipilih adalah jalur yang berlabel No.
	**Simbol Terminator**	Terminator berfungsi untuk menandai awal dan akhir dari suatu flowchart. Simbol ini biasanya diberi label START untuk menandai awal dari flowchart, dan label STOP untuk menandai akhir dari flowchart. Jadi dalam sebuah flowchart pasti terdapat sepasang terminator yaitu terminator start dan stop.

	**Simbol Konektor**	Simbol konektor digunakan pada waktu menghubungkan suatu langkah dengan langkah lain dalam sebuah flowchart dengan keadaan on page atau off page. On page connector digunakan untuk menghubungkan suatu langkah dengan langkah lain dari flowchart dalam satu halaman, sedangkan off page connector digunakan untuk menghubungkan suatu langkah dengan langkah lain dari flowchart dalam halaman yang berbeda. Connector ini biasanya dipakai saat media yang kita gunakan untuk menggambar flowchart tidak cukup luas untuk memuat gambar secara utuh, jadi perlu dipisahpisahkan. Dalam sepasang connector biasanya diberi label tertentu yang sama agar lebih mudah diketahui pasangannya.
	**Simbol Prosedur**	Simbol ini berperan sebagai blok pembangun dari suatu program. Prosedur memiliki suatu flowchart yang berdiri sendiri diluar flowchart utama. Jadi dalam simbol ini, kita cukup menuliskan nama prosedurnya saja, jadi sama seperti jika kita melakukan pemanggilan suatu prosedur pada program utama (main program). Sama dengan aturan pada simbol percabangan, penulisan nama prosedur dilakukan secara satu per satu.

2.5.4 Pengkodean, Uji Coba dan Pembuatan Dokumentasi

Setelah membentuk algoritma, maka proses pengkodean dapat dimulai. Menggunakan algoritma sebagai pedoman, maka kode program dapat ditulis sesuai bahasa pemrograman yang dipilih. Setelah menyelesaikan seluruh kode program, langkah selanjutnya adalah menguji program tersebut apakah telah berfungsi sesuai tujuannya untuk memberikan suatu solusi untuk menyelesaikan suatu masalah. Bilamana terjadi kesalahan – kesalahan logika atas program, disebut juga sebagai *bugs*, maka kita perlu untuk mengkaji ulang rumusan / algoritma yang telah dibuat, kemudian memperbaiki implementasi kode program yang mungkin keliru. Proses ini disebut dengan *debugging*. Terdapat dua tipe kesalahan *(errors)* yang akan dihadapi seorang *programmer*. Yang pertama adalah *compile-time error,* dan yang kedua adalah *runtime error*. *Compile-time errors* muncul jika terdapat kesalahan penulisan kode program. *Compiler* akan mendeteksi kesalahan yang terjadi

sehingga kode tersebut tidak akan bisa dikompilasi.

Terlupakannya penulisan *semi-colon* (;) pada akhir sebuah pernyataan program atau kesalahan ejaan pada beberapa perintah dapat disebut juga sebagai *compile – time error*. *Compiler* tidaklah sempurna sehingga tidak dapat mengidentifikasi seluruh kemungkinan kesalahan pada waktu kompilasi. Umumnya kesalahan yang terjadi adalah kesalahan logika seperti perulangan tak berakhir. Tipe kesalahan ini disebut dengan *runtime error*. Sebagai contoh, penulisan kode pada program terlihat tanpa kesalahan, namun pada saat anda menelusuri struktur logika kode tersebut, bagian yang sama pada kode tereksekusi berulang – ulang tanpa akhir. Pada kasus tersebut *compiler* tidak cukup cerdas untuk menangkap kesalahan tipe ini pada saat proses kompilasi. Sehingga saat program dijalankan, aplikasi atau bahkan keseluruhan komputer mengalami *hang* karena mengalami proses perulangan yang tidak berakhir. Contoh lain dari *run-time errors* adalah perhitungan atas nilai yang salah, kesalahan penetapan kondisi dan lain sebagainya..

Untuk memudahkan dalam memeriksa suatu kesalahan suatu program ataupun memahami jalannya program, kita juga perlu membuat suatu dokumentasi dari program yang dibuat. Dokumentasi tersebut berisi informasi mulai dari tujuan dan fungsi program, algoritma, serta cara penggunaannya.

2.6 Sistem Numerik dan Konversi

Bilangan dapat disajikan dalam beberapa cara. Cara penyajiannya tergantung pada Basis (***BASE***) bilangan tersebut. Terdapat 4 cara utama dalam penyajian bilangan.

2.6.1 Sistem Bilangan Desimal

Manusia umumnya menggunakan bilangan pada bentuk desimal. Bilangan desimal adalah sistem bilangan yang berbasis 10. Hal ini berarti bilangan – bilangan pada sistem ini terdiri dari 0 sampai dengan 9. Berikut ini beberapa contoh bilangan dalam bentuk desimal: 12610 (umumnya hanya ditulis 126), dan 1110 (umumnya hanya ditulis 11).

2.6.2 Sistem Bilangan Biner

Bilangan dalam bentuk biner adalah bilangan berbasis 2. Ini menyatakan bahwa bilangan yang terdapat dalam sistem ini hanya 0 dan 1. Berikut ini contoh penulisan dari bilangan biner :

11111102
10112

1.6.3 Sistem Bilangan Oktal

Bilangan dalam bentuk oktal adalah sistem bilangan yang berbasis 8. Hal ini berarti bilangan – bilangan yang diperbolehkan hanya berkisar antara 0 – 7. Berikut ini contoh penulisan dari bilangan oktal :

1768
138

1.6.4 Sistem Bilangan Heksadesimal

Bilangan dalam sistem heksadesimal adalah sistem bilangan berbasis 16. Sistem ini hanya memperbolehkan penggunaan bilangan dalam skala 0 – 9, dan menggunaan huruf A – F, atau a – f karena perbedaan kapital huruf tidak memiliki efek apapun. Berikut ini contoh penulisan bilangan pada sistem heksadesimal :

7E16
B16

Heksadesimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Nilai Dalam Desimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Tabel 3: Bilangan heksadesimal dan perbandingannya terhadap desimal

Berikut adalah perbandingan keseluruhan sistem penulisan bilangan :

Desimal	*Biner*	*Oktal*	*Heksadesimal*
126_{10}	1111110_2	176_8	$7E_{16}$
11_{10}	1011_2	13_8	B_{16}

Tabel 4: Contoh Konversi Antar Sistem Bilangan

1.6.5 Konversi

1.6.5.1 Desimal ke Biner / Biner ke Desimal

Untuk mengubah angka desimal menjadi angka biner digunakan metode pembagian dengan angka 2 sambil memperhatikan sisanya. Ambil hasil bagi dari proses pembagian sebelumnya, dan bagi kembali bilangan tersebut dengan angka 2. Ulangi langkah – langkah tersebut hingga hasil bagi akhir bernilai 0 atau 1. Kemudian susun nilai – nilai sisa dimulai dari nilai sisa terakhir sehingga diperoleh bentuk biner dari angka bilangan tersebut.

Sebagai Contoh :

$126_{10} = \underline{?_2}$

	Hasil Bagi	*Nilai Sisa*	
126 / 2 =	63	0	↑
63 / 2 =	31	1	
31 / 2 =	15	1	**Urutkan**
15 / 2 =	7	1	
7 / 2 =	3	1	
3 / 2 =	1	1	
1 / 2 =	→	1	

Dengan menuliskan nilai sisa mulai dari bawah ke atas, didapatkan angka biner 1111110**2.** Konversi bilangan biner ke desimal didapatkan dengan menjumlahkan perkalian semua bit biner dengan perpangkatan 2 sesuai dengan posisi bit tersebut.

Sebagai Contoh :

$11001101_2 = \underline{?_{10}}$

* Biner	1	1	0	0	1	1	0	1	11001101
* Desimal	128	64	0	0	8	4	0	1	205
* Pangkat	2^6	2^5	2^5	2^4	2^3	2^2	2^1	2^0	X^{1-6}

Angka desimal 205 diperoleh dari penjumlahan angka yang di arsir. Setiap biner yang bernilai 1 akan mengalami perhitungan, sedangkan yang bernilai 0 tidak akan dihitung karena hanya akan menghasilkan nilai 0.

1.6.5.2 Desimal ke Oktal/Heksadesimal dan Oktal/ Heksadesimal ke Desimal

Pengubahan bilangan desimal ke bilangan oktal atau bilangan heksadesimal pada dasarnya sama dengan konversi bilangan desimal ke biner. Perbedaannya terletak pada bilangan pembagi. Jika pada konversi biner pembaginya adalah angka 2, maka pada konversi oktal pembaginya adalah angka 8, sedangkan pada konversi heksadesimal pembaginya adalah 16.

Contoh konversi Oktal :

$126_{10} = \underline{?_8}$

	Hasil Bagi	***Nilai Sisa***
126 / 8 =	15	6
15 / 8 =	1	7
1 / 8 =	`	1

Dengan menuliskan nilai sisa dari bawah ke atas, kita peroleh bilangan oktal 1768**.** **Contoh konversi Heksadesimal :**

$126_{10} = ?_{16}$

	Hasil Bagi	***Nilai Sisa***
126 / 16 =	7	14 (E)
7 / 16 =		7

Dengan menuliskan nilai sisa dari bawah ke atas, kita peroleh bilangan Heksadesimal 7E**16.** Konversi bilangan Oktal dan Heksadesimal sama dengan konversi bilangan Biner ke Desimal. Perbedaanya hanya terdapat pada penggunaan angka basis. Jika sistem Biner menggunakan basis 2, maka pada bilangan Oktal, basis yang digunakan adalah 8 dan pada bilangan Heksadesimal adalah angka 16.

Contoh konversi Oktal :

$176_8 = ?_{10}$

Posisi	***2***	***1***	***0***	
Octal Digits	**1**	**7**	**6**	
				$6 \times 8^0 = 6$
				$7 \times 8^1 = 56$
				$1 \times 8^2 = 64$
				TOTAL: 126

Contoh konversi Heksadesimal :

$7E_{16} = ?_{10}$

Posisi	***1***	***0***
Digit Heksadesimal	**7**	**E**

$14 \times 16^0 = 14$

$7 \times 16^1 = 112$

TOTAL: 126

1.6.5.3 Biner ke Oktal dan Oktal ke Biner

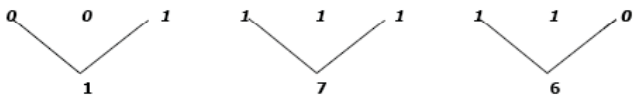
Untuk mengubah bilangan biner ke oktal, kita pilah bilangan tersebut menjadi 3 bit bilangan biner dari kanan ke kiri. Tabel berikut ini menunjukkan representasi bilangan biner terhadap bilangan oktal :

Digit Oktal	*Representasi Biner*
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Tabel 5: Bilangan octal dan perbandingannya dalam sistem biner

Sebagai contoh :

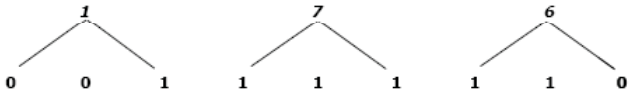
$1111110_2 = ?_8$



Mengubah sistem bilangan oktal menjadi bilangan biner dilakukan dengan cara kebalikan dari konversi biner ke oktal. Dalam hal ini masing – masing digit bilangan oktal diubah langsung menjadi bilangan biner dalam kelompok tiga bit, kemudian merangkai kelompok bit tersebut sesuai urutan semula.

Sebagai contoh :

$176_8 = ?_2$



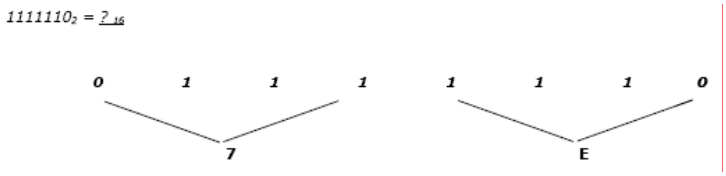
1.6.5.4 Biner ke Heksadesimal dan Heksadesimal ke Biner

Pengubahan bilangan Biner ke Heksadesimal dilakukan dengan pengelompokan setiap empat bit Biner dimulai dari bit paling kanan. Kemudian konversikan setiap kelompok menjadi satu digit Heksadesimal. Tabel berikut menunjukkan representasi bilangan Biner terhadap digit Heksadesimal :

Digit Heksadesimal	*Representasi Biner*
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

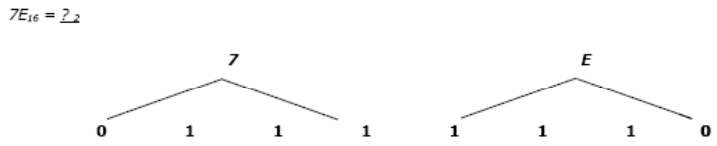
Tabel 6: Bilangan heksadesimal dan konversinya dalam biner

Sebagai contoh :



Konversi bilangan Heksadesimal ke Biner dilakukan dengan membalik urutan dari proses pengubahan Biner ke Heksadesimal. Satu digit Heksadesimal dikonversi menjadi 4 bit Biner.

Sebagai contoh :



1.7 Latihan

1.7.1 Menyusun Algoritma

Dari permasalahan – permasalahan di bawah ini, susunlah sebuah algoritma untuk menyelesaikannya. Anda dapat menyusunnya dengan menggunakan *pseudocode* ataupun *flowchart*.

a. Memasak Roti
b. Menggunakan Komputer di Laboratorium
c. Menghitung rata – rata dari 3 buah bilangan

1.7.2 Konversi Sistem Bilangan

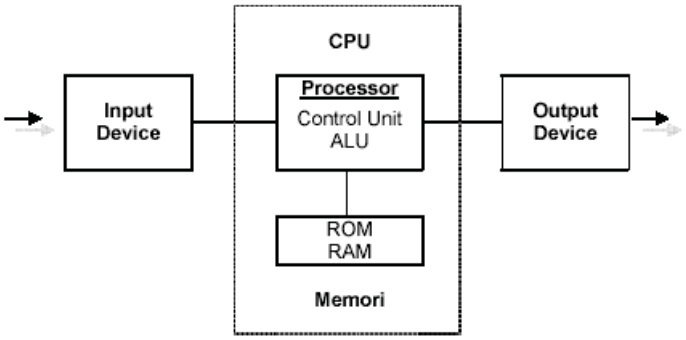
Konversikan bilangan – bilangan berikut ini :

a. 198010 ke sistem bilangan Biner, Heksadesimal dan Oktal
b. 10010011012 ke sistem bilangan Desimal, Heksadesimal dan Oktal
c. 768 ke sistem bilangan Biner, Heksadesimal dan Desimal
d. 43F16 ke sistem bilangan Biner, Desimal dan Oktal.

BAB III
KONSEP DASAR KOMPUTER

3.1 Definisi Komputer

Kata komputer berasal dari bahasa Latin yaitu *Computare* yang artinya *menghitung*. Dalam bahasa Inggris disebut *to compute.* Secara definisi komputer diterjemahkan sebagai sekumpulan alat elektronik yang saling bekerja sama, dapat menerima data (*input*), mengolah data (proses) dan memberikan informasi (*output)* serta terkoordinasi di bawah kontrol program yang tersimpan di memori (lihat gambar)



Prinsip Kerja Komputer

- ***Input Device***, adalah perangkat-perangkat keras komputer yang berfungsi untuk memasukkan data

ke dalam memori komputer, seperti *keyboard*, *mouse*, *joystick* dan lain-lain.

- ***Processor,*** adalah perangkat utama komputer yang mengelola seluruh aktivitas komputer itu sendiri. Prosesor terdiri dari dua bagian utama, yaitu ;
  - ***Control Unit (CU)***, merupakan komponen utama prosesor yang mengontrol semua perangkat yang terpasang pada komputer, mulai dari input device sampai output device.
  - ***Arithmetic Logic Unit (ALU),*** merupakan bagian dari prosesor yang khusus mengolah data aritmatika (menambah, mengurang dll) serta data logika (perbandingan).

- ***Memory*** adalah media penyimpan data pada komputer. Memori ini terbagi atas dua macam, yaitu;
  - ***Read Only Memory (ROM),*** yaitu memori yang hanya bisa dibaca saja, tidak dapat dirubah dan dihapus dan sudah diisi oleh pabrik pembuat komputer. Isi ROM diperlukan pada saat komputer dihidupkan. Perintah yang ada pada ROM sebagian akan dipindahkan ke RAM. Perintah yang ada di ROM antara lain adalah perintah untuk membaca sistem operasi dari disk, perintah untuk mengecek semua peralatan yang ada di unit sistem dan perintah untuk menampilkan pesan di layar. Isi ROM tidak akan hilang meskipun tidak ada aliran listrik. Namun, pada saat sekarang ini ROM telah mengalami perkembangan dan banyak macamnya, antara lain ;

a. ***PROM (Programable ROM),*** yaitu ROM yang bisa kita program kembali dengan catatan hanya boleh satu kali perubahan setelah itu tidak dapat lagi diprogram.

b. ***RPROM (Re-Programable ROM)***, merupakan perkembangan dari versi *PROM* di mana kita dapat melakukan perubahan berulangkali sesuai dengan yang diinginkan.

c. ***EPROM (Erasable Program ROM)***, merupakan *ROM* yang dapat kita hapus dan program kembali, tetapi cara penghapusannya dengan menggunakan sinar ultraviolet.

d. ***EEPROM (Electrically Erasable Program ROM),*** perkembangan mutakhir dari ROM di mana kita dapat mengubah dan menghapus program *ROM* dengan menggunakan teknik elektrik. *EEPROM* ini merupakan jenis yang paling banyak digunakan saat ini.

- ***Random Access Memory (RAM)***, dari namanya kita dapat artikan bahwa *RAM* adalah memori yang dapat diakses secara random. *RAM* berfungsi untuk menyimpan program yang kita olah untuk sementara waktu (*power on*) jika komputer kita matikan, maka seluruh data yang tersimpan dalam *RAM* akan hilang. Tujuan dari *RAM* ini adalah mempercepat pemroses data pada komputer. Agar data yang kita buat tidak dapat hilang pada saat komputer dimatikan, maka diperlukan media penyimpanan eksternal, seperti *Disket*, *Harddisk, PCMCIA Card* dan lain-lain.

- ***Output Device***, adalah perangkat komputer yang berguna untuk menghasilkan keluaran, apakah itu ke kertas (*hardcopy*), ke layar monitor (*softcopy*) atau keluaran berupa suara. Contohnya printer, speaker, plotter, monitor dan banyak yang lainnya.

Dari penjelasan di atas dapat kita simpulkan bahwa **prinsip kerja komputer** tersebut diawali memasukan data dari **perangkat *input***, lalu data tersebut diolah sedemikian rupa oleh ***CPU*** sesuai yang kita inginkan dan data yang telah diolah tadi disimpan dalam **memori** komputer atau disk. Data yang disimpan dapat kita lihat hasilnya melalui perangkat keluaran.

3.2 Komponen – Komponen Komputer

Komputer terdiri atas tiga komponen utama yang tidak dapat dipisahkan, yaitu;

- ***Hardware*** (perangkat keras), merupakan peralatan fisik dari komputer yang dapat kita lihat dan rasakan. Hardware ini terdiri atas
  - ***Input/Output Device (I/O Device)***
  - Terdiri atas perangkat masukan dan keluaran, seperti *keyboard* dan *printer*.
  - ***Storage Device*** (perangkat penyimpanan) Merupakan media untuk menyimpan data seperti *disket, harddisk, CD-I*,dll.
  - Monitor /*Screen*
  - Monitor merupakan sarana untuk menampilkan apa yang kita ketikkan pada papan *keyboard* setelah diolah oleh prosesor. Monitor disebut juga dengan *Visual Display Unit* (*VDU*).
  - ***Casing Unit***

- Casing unit adalah tempat dari semua peralatan komputer, baik itu *motherboard*, *card, peripheral* lain dan *Central Procesing Unit (CPU)*. *Casing* unit ini disebut juga dengan System Unit.
- ***Central Procesing Unit*** (*CPU*)
- *Central Procesing Unit* adalah salah satu bagian komputer yang paling penting, karena jenis prosesor menentukan pula jenis komputer. Baik tidaknya suatu komputer, jenis komputer, harga komputer, ditentukan terutama oleh jenis prosesornya. Semakin canggih prosesor komputer, maka kemampuannya akan semakin baik dan biasanya harganya akan semakin mahal.

- ***Software*** (perangkat lunak), merupakan program-program komputer yang berguna untuk menjalankan suatu pekerjaan sesuai dengan yang dikehendaki. Program tersebut ditulis dengan bahasa khusus yang dimengerti oleh komputer. Software terdiri dari beberapa jenis, yaitu ;
  - **Sistem Operasi,** seperti *DOS, Unix, Novell, OS/2, Windows*, dll. Adalah software yang berfungsi untuk mengaktifkan seluruh perangkat yang terpasang pada komputer sehingga masing-masingnya dapat saling berkomunikasi. Tanpa ada sistem operasi maka komputer tak dapat difungsikan sama sekali.
  - ***Program Utility***, seperti *Norton Utility, Scandisk, PC Tools*, dll. Program utility berfungsi untuk membantu atau mengisi kekurangan/ kelemahan dari sistem operasi, misalnya *PC Tools* dapat melakukan perintah format

sebagaimana *DOS*, tapi *PC Tools* mampu memberikan keterang dan animasi yang bagus dalam proses pemformatan. File yang telah dihapus oleh *DOS* tidak dapat dikembalikan lagi, tetapi dengan program bantu hal ini dapat dilakukan.

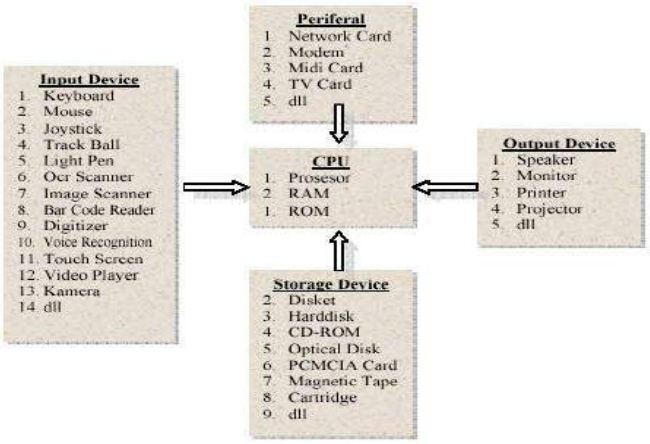
- **Program Aplikasi**, seperti *GL, MYOB, Payroll*, dll. Merupakan program yang khusus melakukan suatu pekerjaan tertentu, seperti program gaji pada suatu perusahaan. Program ini hanya digunakan oleh bagian keuangan saja tidak dapat digunakan oleh departemen yang lain. Biasanya program aplikasi ini dibuat oleh seorang *programmer* komputer sesuai dengan permintaan/kebutuhan seseorang/lembaga/ perusahaan guna keperluan interennya.
- **Program Paket**, seperti *MS-Word, MS-Excel, Lotus 125*, dll Adalah program yang disusun sedemikian rupa sehingga dapat digunakan oleh banyak orang dengan berbagai kepentingan. Seperti *MS-Word*, dapat digunakan oleh departemen keuangan untuk membuat nota, atau bagian administrasi untuk membuat surat penawaran dan lain sebagainya.
- **Bahasa Pemrograman**, *Pascal, Fortran, Clipper, dBase*, dll. Merupakan *software* yang khusus digunakan untuk membuat program komputer, apakah itu sistem operasi, program paket dll. Bahasa pemrograman ini biasanya dibagi atas 3 tingkatan, yaitu ;
  - ***a.*** ***Low Level Language***, bahasa pemrograman generasi pertama, bahasa pemrograman jenis ini sangat sulit dimengerti karena

instruksinya menggunakan bahasa mesin. Biasanya yang mengerti hanyalah pembuatnya saja.

b. ***Midle Level Language***, merupakan bahasa pemrograman tingkat menengah di mana penggunaan instruksi sudah mendekati bahasa sehari-hari, walaupun begitu masih sulit untuk dimengerti karena banyak menggunakan singkatan-singakatan seperti STO artinya simpan (singkatan dari STORE) dan MOV artinya pindah (singkatan dari MOVE).Yang tergolong kedalam bahasa ini adalah *Assembler*, *ForTran (Formula Translator).*

c. ***High Level Language***, merupakan bahasa tingkat tinggi yang mempunyai cirri mudah dimengerti, karena menggunakan bahasa sehari-hari, seperti *BASIC, COBOL, dBase* dll.

- **Brainware** (*user*), adalah personil-personil yang terlibat langsung dalam pemakaian komputer, seperti Sistem analis, *programmer, operator, user*, dll. Pada organisasi yang cukup besar, masalah komputerisasi biasanya ditangani oleh bagian khusus yang dikenal dengan bagian *EDP (Electronic Data Processing),* atau sering disebut dengan *EDP* Departemen, yang dikepalai oleh seorang Manager *EDP*.



Komponen Perangkat Keras Komputer

3.3 Pengelompokan Komputer

Beberapa tahun lalu, penggolongan komputer dilakukan atas dasar besarnya RAM yang ada pada tiap komputer. Waktu itu, komputer yang memiliki memori atau RAM antara 512 KB hingga 1 MB disebut dengan **Komputer Mikro** dan yang memiliki RAM lebih dari 1 MB disebut **Komputer Mini.** Penggolongan seperti ini sekarang tidak tepat lagi, karena komputer sakupun sekarang sudah banyak yang memiliki RAM lebih besar dari 1 MB. (1 MB = 1.024 KB).

Penggolongan jenis-jenis komputer yang lebih tepat adalah berdasarkan jenis prosesor yang ada pada komputer, karena kemampuan kerja komputer ditentukan oleh kemampuan prosesornya, semakin tinggi jenis prosesor yang digunakan, maka semakin tinggi pulalah kinerja dari komputer tersebut. Penggolongan komputer berdasarkan kriteria lain masih dimungkinkan, misalnya berdasarkan ukuran fisik, sistem operasi, dan jenis data yang diolah (lihat tabel).

DASAR PENGGOLONGAN	JENIS KOMPUTER
Generasi	*I (1946 - 1959)* *II (1960 - 1965)* *III (1966 - 1970)* *IV (1971 - sekarang)* *V (pemanfaat teknologi Artificial Inteligence (AI))*
Prosesor	*Mainframe* *Minikomputer* *IBM compatible atau PC* *Macintosh*
Bentuk dan ukuran Fisik	*Tower* *Desktop* *Portable* *Laptop* *Komputer* *Notebook* *Sub notebook (handbook)* *Palmtop (handheld)*
Sistem Operasi	*Unix* *Netware* *DOS* *OS/2* *Windosw NT (Network Technology)* *System* *NextStep*
Jenis data yang diolah	*Analog* *Digital* *Hybrid*

1. Jenis Komputer Berdasarkan Prosesor

Berdasarkan prosesornya, komputer digolongkan ke dalam tiga bagian, yaitu *Mainframe, Mini Computer dan Personal Computer* (PC). Penggolongan ini dalam beberapa tahun mendatang akan semakin kabur dan mungkin akan hilang, karena komputer mainframe dan mini mengalami perkembangan yang lambat, sementara komputer PC berkembang terus dengan pesatnya.

- ***Mainframe*** adalah komputer yang prosesornya mempunyai kemampuan sangat besar, karena ditujukan untuk banyak pemakai. *Mainframe* menyediakan sedikit waktu dan sebagian memorinya untuk setiap pemakai (*user*), kemudian berpindah lagi kepada pemakain lain, lalu kembali kepemakai yang pertama. Perpindahan ini tidak dirasakan oleh pemakai, seolah-olah tidak ada apa-apa. *Mainframe* disediakan untuk banyak pemakai (*multi user*) dan setiap pemakai dapat menggunakan program yang berbeda pada saat yang sama (*multitasking*). Komputer *mainframe* mempunyai CPU yang

berada pada satu mesin sendiri, mempunyai perangkat penyimpanan, komunikasi di satu mesin sendiri dan dihubungkan dengan banyak terminal yang terdiri dari *keyboard* dan *monitor* saja. Komputer jenis ini biasanya digunakan pada perusahaan yang berskala besar, seperti kantor pusat penerbangan nasional. Komputer *mainframe* saat sekarang kalah saing dengan komputer PC dengan teknologi internet.

- ***Mini Computer*** sebenarnya adalah bentuk mini dari komputer *mainframe*. Kalau *mainframe* dapat memiliki ribuan terminal, komputer mini lebih terbatas hanya sampai puluhan dan mungkin hanya ratusan. Komputer mini ditujukan untuk perusahaan yang tidak begitu besar tetapi juga tidak begitu kecil. Komputer mini cocok untuk perguruan tinggi yang hanya memiliki satu atau dua fakultas, pabrik yang produknya hanya untuk memenuhi kebutuhan daerah setempat. Komputer mini ini sekarang jarang dipakai, karena lebih fleksibel menggunakan komputer PC dengan teknologi *Local Area Networknya (LAN)*

- ***Personal Computer (PC)*** atau komputer pribadi adalah komputer yang ditujukan untuk satu pemakai dengan satu pemakain program aplikasi pada suatu saat. Oleh karenanya, perangkatnya dapat diringkas ke dalam satu mesin saja. Komputer ini memiliki monitor, *keyboard* dan CPU. Namun, di dalam CPU ini sebenarnya tidak hanya terdapat prosesor saja, tetapi juga ada perangkat penyimpanan dan mungkin saja dipasangi perangkat tambahan (periferal). Komputer jenis inilah yang paling banyak digunakan, baik itu di rumah, kantor, lembaga kursus, sekolah dll. Dengan menambahkan berbagai perangkat tambahan, komputer PC dapat menandingi komputer *mainframe* dan mini, seperti telah dijelaskan di atas.

2. Jenis Komputer Berdasarkan Bentuk dan Ukuran Fisik

   Perlu diketahui bahwa komputer tidak dibedakan kemampuannya berdasarkan ukuran fisiknya. Bukan berarti komputer yang kecil bentuknya berarti kecil pula kemampuannya.

   a. *Tower* (menara) adalah yang biasanya diletakkan disamping atau di bawah meja, karena ukurannya yang relatif besar, sehingga memenuhi meja. Komputer ini biasanya banyak

memiliki ruang di dalamnya dan banyak memiliki *expansion slot* (tempat untuk memasang *card* tambahan), sehingga bisa ditambahkan dengan berbagai perangkat tambahan.

b. *Desktop* (meja) adalah komputer yang ukuran sedikit lebih kecil dari dari *Tower*, tetapi biasanya diletakkan diatas meja. Komputer ini paling banyak dipakai karena harganya yang lebih murah bila dibandingkan dengan bentuk yang lain. Komputer yang kita pakai sekarang ini adalah jenis *desktop*.

c. *Portable* (mudah di bawah-bawah) adalah komputer yang ukuran sedikit lebih kecil dari *Desktop*, karena bagian-bagiannya dapat dirangkai menjadi satu kotak saja, sehingga mudah dibawa ke mana-mana. Komputer ini ditujukan bagi pemakai yang sering bertugas dilapangan, misalnya insinyur yang bertugas menyelesaikan suatu rumah atau peneliti yang mengumpulkan data di lokasi yang jauh dari kantornya. Komputer ini kurang popular karena relatif besar dan berat.

d. *Notebook* (buku catatan) adalah komputer yang ukurannya sebesar buku catatan (yang banyak dipakai pelajar dan mahasiswa Amerika) saja. *Notebook* mempunyai ukuran yang sama dengan kertas kwarto, yaitu 8 ½ x 11 inci, tebalnya berkisar 1 hinggan 1 ½ inci dan beratnya antara 4 sampai 6 kg.

e. *Subnotebook* adalah komputer yang ukuran ada di antara komputer *notebook* dan *palmtop*. Ukuran komputer ini sedikit lebih kecil dari notebook karena ada sebagian perangkat yang tidak dipasang, biasanya *disk drive*.

f. *Palmtop* adalah komputer yang dapat digenggam, karena ukurannya yang sangat kecil, kira-kira sedikit lebih kecil dibandingkan kaset video Beta. Komputer ini sering disebut *handheld computer*. Komputer ini tidak memerlukan aliran listrik, melainkan baterai kecil biasa (ukuran AA). Kelemahan dari komputer ini adalah layarnya yang terlalu kecil dan *keyboard*-nya sedikit lebih kecil dari ukuran standar, sehingga menyulitkan pamakai.

3. Jenis Komputer Berdasarkan Data yang Diolah

Berdasarkan pada data yang diolahnya, komputer dapat dibagai atas tiga bagian, yaitu ;

a. **Komputer Analog** digunakan untuk mengolah data kualitatif, bekerja secara kontinyu dan parallel, biasanya tidak memerlukan bahasa perantara. Contohnya komputer yang digunakan di rumah sakit untuk mengukur suhu, kecepatan suara, voltase listrik dll.

b. **Komputer Digital** digunakan untuk mengolah data kuantitatif (huruf, angka, kombinasi huruf & angka, karakter-karakter khusus) biasanya memerlukan bahasa perantara. Contohnya komputer PC dll.

c. **Komputer Hybrid**, merupakan kombinasi antara komputer analog dengan digital. Contohnya *Facsimile*.

BAB IV
SISTEM OPERASI KOMPUTER

4.1 Definisi Sistem Operasi

Sistem operasi merupakan salah satu komponen utama dari sebuah sistem komputer. Komponen – komponen tersebut, yaitu *hardware* (perangkat keras), *software* (perangkat lunak aplikasi), dan *brainware* (para pengguna). Di mana sistem operasi tersebut merupakan bagian dari s*oftware* (lihat kembali pembahasan komponen – komponen komputer di atas).

Secara lebih rinci, sistem operasi didefinisikan sebagai sebuah program yang mengatur perangkat keras komputer, serta bertindak sebagai penghubung antara para pengguna dengan perangkat keras. Sistem operasi bertugas untuk mengendalikan (kontrol) serta mengkoordinasikan pengunaan perangkat keras untuk berbagai program aplikasi untuk bermacam-macam pengguna.

Sejarah Perkembangan Sistem Operasi

Pada awalnya, komputer berukuran sangat besar sehingga komponen – komponennya dapat memenuhi sebuah ruangan yang sangat besar. Pengguna menjadi *programmer* yang sekaligus merangkap menjadi *operator*

komputer, juga bekerja di dalam ruang komputer tersebut.. Walau pun berukuran besar, sistem tersebut diketegorikan sebagai komputer pribadi (PC). Siapa saja yang ingin melakukan komputasi, harus memesan/antri untuk mendapatkan alokasi waktu (rata-rata 30-120 menit). Jika ingin melakukan kompilasi *Fortran*, maka pengguna pertama kali akan me-*load* kompilator *Fortran*, yang diikuti dengan "*load*" program dan data. Hasil yang diperoleh, biasanya berbentuk cetakan (*print-out*).

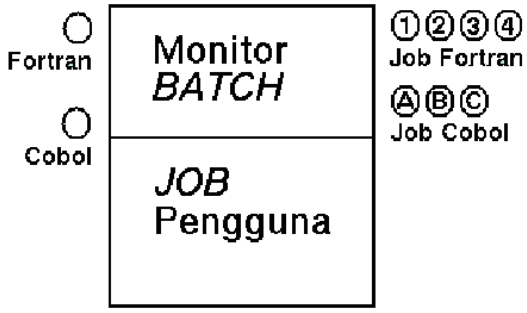
Timbul beberapa masalah pada sistem PC tersebut. Umpama, alokasi pesanan harus dilakukan di muka. Jika pekerjaan rampung sebelum rencana semula, maka sistem komputer menjadi *idle*/tidak tergunakan. Sebaliknya, jika perkerjaan rampung lebih lama dari rencana semula, para calon pengguna berikutnya harus menunggu hingga pekerjaan selesai. Selain itu, seorang pengguna kompilator *Fortran* akan beruntung, jika pengguna sebelumnya juga menggunakan *Fortran*. Namun, jika pengguna sebelumnya menggunakan *Cobol*, maka pengguna *Fortran* harus me-*load*. Masalah ini ditanggulangi dengan menggabungkan para pengguna kompilator sejenis ke dalam satu kelompok *batch* yang sama. Medium semula yaitu *punch card* diganti dengan *tape*.

Selanjutnya, terjadi pemisahan tugas antara *programmer* dan *operator*. Para operator biasanya secara eksklusif menjadi penghuni ruang kaca seberang ruang komputer. Para *programmer* yang merupakan pengguna (*users*), mengakses komputer secara tidak langsung melalui bantuan para *operator*. Para pengguna mempersiapkan sebuah *job* yang terdiri dari program aplikasi, data masukan, serta beberapa perintah pengendali program.

Medium yang lazim digunakan ialah kartu berlubang (*punch card*). Setiap kartu dapat menampung informasi

satu baris hingga 80 karakter. Set kartu *job* lengkap tersebut kemudian diserahkan kepada para operator.

Perkembangan sistem operasi dimulai dari sini, dengan memanfaatkan sistem *batch* (lihat gambar sistem monitor *batch* sederhana). Para *operator* mengumpulkan *job-job* yang mirip yang kemudian dijalankan secara berkelompok. Umpama, *job* yang memerlukan kompilator *Fortran* akan dikumpulkan ke dalam sebuah *batch* bersama dengan *job-job* lainnya yang juga memerlukan kompilator *Fortran*. Setelah sebuah kelompok *job* rampung, maka kelompok *job* berikutnya akan dijalankan secara otomatis.



Sistem Monitor *Batch* Sederhana

Pada perkembangan berikutnya, diperkenalkan konsep *Multiprogrammed System*. Dengan sistem ini, *job-job* disimpan di main memory pada waktu yang sama dan *CPU* dipergunakan bergantian. Hal ini membutuhkan beberapa kemampuan tambahan, yaitu penyediaan *I/O routine* oleh sistem, pengaturan memori untuk mengalokasikan *memory* pada beberapa *Job*, penjadualan *CPU* untuk memilih *job* mana yang akan dijalankan, serta pengalokasian perangkat keras lain (lihat gambar model *multyprogram system*).

Peningkatan lanjut dikenal sistem bagi waktu/ tugas ganda/komputasi interaktif (*Time-Sharing System*/

Multitasking/Interactive Computing). Sistem ini, secara simultan dapat diakses lebih dari satu pengguna. *CPU* digunakan bergantian oleh *job-job* di memori dan di disk. *CPU* dialokasikan hanya pada *job* di memori dan *job* dipindahkan dari dan ke disk. Interaksi langsung antara pengguna dan komputer ini melahirkan konsep baru, yaitu *response time* yang diupayakan wajar agar tidak terlalu lama menunggu.

Hingga akhir tahun 1980-an, sistem komputer dengan kemampuan yang normal, lazim dikenal dengan istilah *main-frame*. Sistem komputer dengan kemampuan jauh lebih rendah (dan lebih murah) disebut komputer mini. Sebaliknya, komputer dengan kemampuan jauh lebih canggih disebut komputer super (*super-computer*). CDC 6600 merupakan yang pertama dikenal dengan sebutan komputer super menjelang akhir tahun 1960-an. Namun, prinsip kerja sistem operasi dari semua komputer tersebut lebih kurang sama saja.

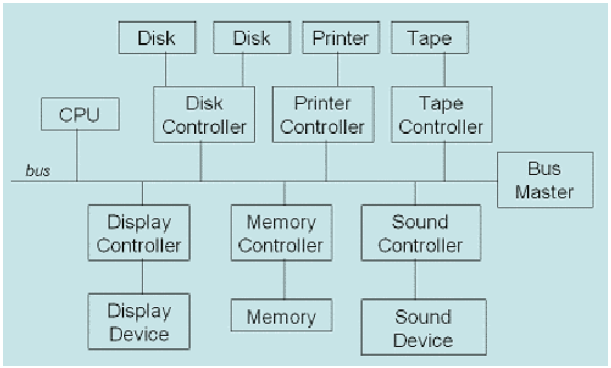
Komputer klasik seperti diungkapkan di atas, hanya memiliki satu prosesor. Keuntungan dari sistem ini ialah lebih mudah diimplementasikan karena tidak perlu memperhatikan sinkronisasi antarprosesor, kemudahan kontrol terhadap prosesor karena sistem proteksi tidak, teralu rumit, dan cenderung murah (bukan ekonomis). Perlu dicatat yang dimaksud satu buah prosesor ini ialah satu buah prosesor sebagai *Central Processing Unit* (CPU). Hal ini ditekankan sebab ada beberapa perangkat yang memang memiliki prosesor tersendiri di dalam perangkatnya seperti *VGA Card AGP*, *Optical Mouse*, dan lain-lain.

Sistem Operasi
JOB1
JOB2
JOB3
JOB4
JOB5
JOB6

Model *Multyprogram System*

4.2 Operasi Sistem Komputer

Secara umum, sistem komputer terdiri atas CPU dan sejumlah *device controller* yang terhubung melalui sebuah *bus* yang menyediakan akses ke memori. Umumnya, setiap *device controller* bertanggung jawab atas sebuah *hardware* spesisfik. Setiap *device* dan CPU dapat beroperasi secara konkuren untuk mendapatkan akses ke memori. Adanya beberapa *hardware* ini dapat menyebabkan masalah sinkronisasi. Karena itu, untuk mencegahnya sebuah *memory controller* ditambahkan untuk sinkronisasi akses memori.



Arsitektur Komputer Umum

Pada sistem komputer yang lebih maju, arsitekturnya lebih kompleks. Untuk meningkatkan performa, digunakan beberapa buah *bus*. Tiap *bus* merupakan jalur data antara beberapa *device* yang berbeda. Dengan cara ini, *RAM, Prosesor, GPU (VGA AGP)* dihubungkan oleh *bus* utama berkecepatan tinggi yang lebih dikenal dengan nama *FSB (Front Side Bus)*. Sementara perangkat lain yang lebih lambat dihubungkan oleh *bus* yang berkecepatan lebih rendah yang terhubung dengan *bus* lain yang lebih cepat sampai ke bus utama. Untuk komunikasi antarbus ini digunakan sebuah *bridge*.

Tanggung jawab sinkronisasi *bus* yang secara tak langsung juga mempengaruhi sinkronisasi memori dilakukan oleh sebuah *bus controller* atau dikenal sebagai *bus master*. *Bus master* akan mengendalikan aliran data hingga pada satu waktu, bus hanya berisi data dari satu buah *device*. Pada prakteknya *bridge* dan *bus master* ini disatukan dalam sebuah *chipset*.

Jika komputer dinyalakan, yang dikenal dengan nama *booting*, komputer akan menjalankan *bootstrap program* yaitu sebuah program sederhana yang disimpan dalam ROM yang berbentuk chip *CMOS (Complementary Metal Oxide Semiconductor)*. Chip CMOS modern biasanya bertipe *Electrically Erasable Programmable Read Only Memory* (EEPROM), yaitu memori *non-volatile* (tak terhapus jika power dimatikan) yang dapat ditulis dan dihapus dengan pulsa elektronik. Lalu *bootsrap program* ini lebih dikenal sebagai *BIOS (Basic Input Output System)*.

Bootstrap program utama, yang biasanya terletak di *motherboard* akan memeriksa perangkat keras utama dan melakukan inisialisasi terhadap program dalam *hardware* yang dikenal dengan nama *firmware*.

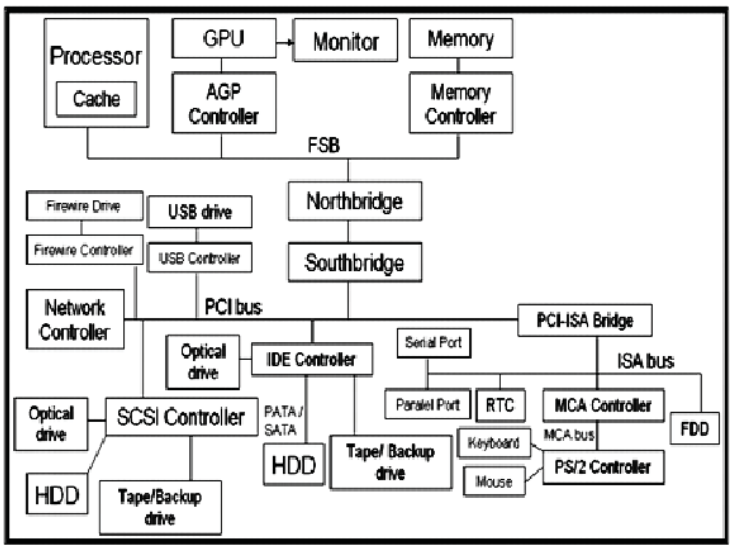
Bootstrap program utama kemudian akan mencari dan

meload *kernel* sistem operasi ke memori lalu dilanjutkan dengan inisialisasi sistem operasi. Dari sini program sistem operasi akan menunggu kejadian tertentu. Kejadian ini akan menentukan apa yang akan dilakukan sistem operasi berikutnya (*event-driven*).

Kejadian ini pada komputer modern biasanya ditandai dengan munculnya *interrupt* dari software atau hardware, sehingga Sistem Operasi ini disebut *Interrupt-driven. Interrupt* dari *hardware* biasanya dikirimkan melalui suatu signal tertentu, sedangkan *software* mengirim *interrupt* dengan cara menjalankan *system call* atau juga dikenal dengan istilah *monitor call. System/Monitor call* ini akan menyebabkan *trap* yaitu *interrupt* khusus yang dihasilkan oleh software karena adanya masalah atau permintaan terhadap layanan sistem operasi. *Trap* ini juga sering disebut sebagai *exception*.

Setiap *interrupt* terjadi, sekumpulan kode yang dikenal sebagai *ISR (Interrupt Service Routine)* akan menentukan tindakan yang akan diambil. Untuk menentukan tindakan yang harus dilakukan, dapat dilakukan dengan dua cara yaitu *polling* yang membuat komputer memeriksa satu demi satu perangkat yang ada untuk menyelidiki sumber *interrupt* dan dengan cara menggunakan alamat-alamat *ISR* yang disimpan dalam array yang dikenal sebagai *interrupt vector* di mana system akan memeriksa *Interrupt Vector* setiap kali *interrupt* terjadi.

Arsitektur *interrupt* harus mampu untuk menyimpan alamat instruksi yang di- *interrupt*. Pada komputer lama, alamat ini disimpan di tempat tertentu yang tetap, sedangkan padakomputer baru, alamat itu disimpan di *stack* bersama-sama dengan informasi state saat itu.



Arsitektur Komputer Modern

BAB V
ALGORITMA PEMROGRAMAN

5.1 Apakah Itu Algoritma

Ditinjau dari asal usulnya, kata Algoritma sendiri mempunyai sejarah yang aneh. Orang hanya menemukan kata *Algorism* yang berarti proses menghitung dengan angka arab. Anda dikatakan *Algorist* jika Anda menghitung menggunakan Angka Arab. Para ahli bahasa berusaha menemukan asal kata ini, tetapi hasilnya kurang memuaskan. Akhirnya, para ahli sejarah matematika menemukan asal kata tersebut yang berasal dari nama penulis buku arab yang terkenal, yaitu Abu Ja'far Muhammad Ibnu Musa Al-Khuwarizmi. Al-Khuwarizmi dibaca orang barat menjadi *Algorism.* Al-Khuwarizmi menulis buku yang berjudul *Kitab Al Jabar Wal-Muqabala* yang artinya "Buku pemugaran dan pengurangan" (*The book of restoration and reduction*). Dari judul buku itu, kita juga memperoleh akar kata "Aljabar" (*Algebra*). Perubahan kata dari *Algorism* menjadi *Algorithm* muncul karena kata *Algorism* sering dikelirukan dengan *Arithmetic*, sehingga akhiran *–sm* berubah menjadi *–thm.* Karena perhitungan dengan angka Arab sudah menjadi hal yang biasa, maka lambat laun kata *Algorithm* berangsur-angsur dipakai sebagai metode perhitungan (komputasi) secara umum,

sehingga kehilangan makna kata aslinya. Dalam Bahasa Indonesia, kata *Algorithm* diserap menjadi *Algoritma*.

5.2 Definisi Algoritma

"Algoritma adalah urutan langkah-langkah logis penyelesaian masalah yang disusun secara sistematis dan logis". Kata *Logis* merupakan kata kunci dalam Algoritma. Langkah-langkah dalam Algoritma harus logis dan harus dapat ditentukan bernilai salah atau benar.

Algoritma Merupakan Jantung Ilmu Informatika Algoritma adalah jantung ilmu komputer atau informatika. Banyak cabang ilmu komputer yang diacu dalam terminologi algoritma. Namun, jangan beranggapan algoritma selalu identik dengan ilmu komputer saja. Dalam kehidupan sehari-haripun banyak terdapat proses yang dinyatakan dalam suatu algoritma. Cara-cara membuat kue atau masakan yang dinyatakan dalam suatu resep juga dapat disebut sebagai algoritma. Pada setiap resep selalu ada urutan langkah-lankah membuat masakan. Bila langkah-langkahnya tidak logis, tidak dapat dihasilkan masakan yang diinginkan. Ibu-ibu yang mencoba suatu resep masakan akan membaca satu per satu langkah-langkah pembuatannya lalu ia mengerjakan proses sesuai yang ia baca. Secara umum, pihak (benda) yang mengerjakan proses disebut pemroses (*processor*). Pemroses tersebut dapat berupa manusia, komputer, robot atau alat-alat elektronik lainnya. Pemroses melakukan suatu proses dengan melaksanakan atau "mengeksekusi" algoritma yang menjabarkan proses tersebut.

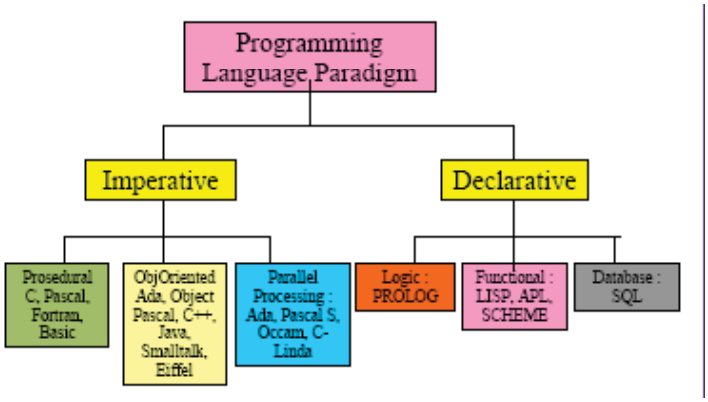
Melaksanakan Algoritma berarti mengerjakan langkah-langkah di dalam Algoritma tersebut. Pemroses mengerjakan proses sesuai dengan algoritma yang diberikan

kepadanya. Juru masak membuat kue berdasarkan resep yang diberikan kepadanya, pianis memainkan lagu berdasarkan papan not balok. Karena itu suatu Algoritma harus dinyatakan dalam bentuk yang dapat dimengerti oleh pemroses. Jadi suatu pemroses harus :

- Mengerti setiap langkah dalam Algoritma
- Mengerjakan operasi yang bersesuaian dengan langkah tersebut.

Mekanisme pelaksanan algoritma oleh pemroses komputer hanyalah salah satu pemroses. Agar dapat dilaksanakan oleh komputer, algoritma hasrus ditulis dalam notasi bahasa pemrograman sehingga dinamakan program.

Jadi, program adalah prwujudan atau implementasi teknis Algoritma yang ditulis dalam bahasa pemrogaman tertentu sehingga dapat dilaksanakan oleh komputer. Belajar Memprogram dan Belajar Bahasa Pemrograman tidak sama. Belajar memprogram adalah belajar tentang metodologi pemecahan masalah, kemudian menuangkannya dalam suatu notasi tertentu yang mudah dibaca dan dipahami. Sedang akan belajar bahasa pemrograman berarti belajar memakai suatu bahasa aturan-aturan tata bahasanya, instruksi-instruksinya, tata cara pengoperasian *compiler*-nya, dan memanfaatkan instruksi-instruksi tersebut untuk membuat program yang ditulis hanya dalam bahasa itu saja.



Sampai saat ini terdapat puluhan bahasa pemrogram. Yang dapat dibedakan berdasarkan tujuan dan fungsinya. Di antaranya adalah:

Belajar Memprogram

- Belajar memprogram ≠ belajar bahasa pemrograman
- Belajar memprogram : belajar tentang strategi pemecahan masalah, metodologi dan sistematika pemecahan masalah kemudian menuliskannya dalam notasi yang disepakati bersama.
- Belajar memprogram : bersifat pemahaman persoalan, analisis dan sintesis
- Belajar memprogram, titik berat : designer program Belajar Bahasa Pemrograman.

Belajar bahasa pemrograman :

- Belajar memakai suatu bahasa pemrograman, aturan sintaksis, tatacara untuk memanfaatkan instruksi yang spesifik untuk setiap bahasa.
- Belajar bahasa pemrograman , titik berat : *coder*

Produk yang dihasilkan pemrogram :

- Program dengan rancangan yang baik (metodologis, sistematis.)
- Dapat dieksekusi oleh mesin.
- Berfungsi dengan benar.
- Sanggup melayani segala kemungkinan masukan.
- Disertai dokumentasi.
- Belajar memprogram, titik berat : designer program.

Algoritma Aksi :

– kejadian yang terjadi pada selang waktu terbatas (dimulai saat T0 dan berakhir pada saat T1).

– Menghasilkan efek netto yang terdefinisi dengan baik dan direncanakan.

Contoh :

a. Ibu Tati mengupas kentang untuk mempersiapkan makan malam..(luas ruang lingkupnya).
b. Karena ruang lingkup luas, maka harus didefinisikan keadaan awal dan efek netto yang direncanakan (Initial State dan Final State).
c. Initial State (keadaan awal) : T0 kentang sudah ada dikantong kentang, dan ditaruh di rak dapur dimana ibu Tati akan mengupasnya.
d. Final State (keadaan akhir) : T1 kentang dalam keadaan terkupas di panci, siap untuk dimasak dan kantong kertasnya harus dikembalikan ke rak lagi.
e. Kejadian : urut-urutan dari beberapa aksi yang terjadi secara berurutan.
f. Efek kumulatif dari semua aksi yang terjadi menjadi efek netto dari kejadian
g. Penggolongan suatu kejadian menjadi aksi

adalah relatif tergantung dari sudut pandang.

- Contoh mengupas kentang dapat dijelaskan :
- Ambil kantong kentang dari rak.
- Ambil panci dari almari.
- Kupas kentang.
- Kembalikan kantong kentang ke rak.
- Contoh lain (jika tidak dipandang perlu untuk menjelaskan kantong kentang diambil dari rak sebelum ambil panci) :
- Ambil kantong kentang dari rak dan ambil panci dari almari.
- Kupas kentang.
- Kembalikan kantong kentang ke rak.

Jika esok hari ibu Tati mengupas kentang lagi untuk makan malam juga, dan kita mengamati hal-hal yang sama, apakah hal tsb bisa disebut sama ? Ini tergantung jawabannya bisa sama bisa tidak. Tidak karena ibu Tati tidak mungkin mengupas kentang yang sama dengan kemarin Sama karena kemiripan pola yang dilakukan.

Notasi Algoritma Independen terhadap Bahasa Pemrograman dan Mesin Komputer. Notasi Algoritma dapat diterjemahkan ke dalam berbagai bahasa pemrograman. Analoginya sama dengan resep membuat kue. Sebuah resep dapat ditulis dalam bahasa apa pun. Bahasa Jepang, Inggris, Perancis, Indonesia, dan lain sebagainya. Apa pun bahasanya, kue yang dihasilkan tetap sama asalkan semua aturan pada resep diikuti. Mengapa demikian ? Karena setiap juru masak (sebagai pemroses) dapat melakukan operasi dasar yang sama, seperti mengocok telur, menimbang berat gula, dan lain sebagainya.

Demikian juga halnya dengan komputer. Meskipun

setiap komputer berbeda teknologinya, tetapi secara umum semua komputer dapat melakukan operasi-operasi dasar dalam pemrograman seperti operasi pembacaan data, operasi perbandingan, operasi aritmatika, dan sebagainya. Perkembangan teknologi komputer tidak mengubah operasi-operasi dasar it, yang berubah hanyalah kecepatan, biaya, atau tingkat ketelitian. Pada sisi lain setiap program dalam bahasa tingkat tinggi selalu diterjemahkan kedalam bahasa mesin sebelum akhirnya dikerjakan oleh CPU. Setiap instruksi dalam bahasa mesin menyajikan operasi dasar yang sesuai, dan menghasilkan efek netto yang sama pada setiap komputer.

5.3 Pemrograman Prosedural

Algoritma berisi urutan langkah-langkah penyelesaian masalah. Ini berarti Algoritma adalah proses yang procedural.

Definisi Prosedural menurut Kamus Besar Bahasa Indonesia :

1. Tahap-tahap kegiatan untuk menyelesaikan suatu aktivitas.
2. Metode langkah demi langkah secara eksak dalam memecahkan suatu masalah.

Pada pemrograman prosedural, program dibedakan antara bagian data dengan bagian instruksi. Bagian instruksi terdiri atas runtutan (*sequence*) instruksi yang dilaksanakan satu per satu secara berurutan oleh pemroses. Alur pelaksanaan instruksi dapat berubah, karena adanya pencabangan kondisional. Data yang disimpan di dalam memori dimanipulasi oleh instrusi secara beruntun atau prosedural. Paradigma pemrograman seperti ini

dinamakan pemrograman prosedural. Bahasa-bahasa tingkat tinggi seperti *Cobol, Basic, Pascal, Fortran* dan *C* mendukung kegiatan pemrograman prosedural, karena itu mereka dinamakan juga bahasa prosedural.

Selain paradigma pemrograman prosedural, ada lagi paradigma yang lain yaitu pemrograman berorientasi objek (*Object Oriented Programming)*. Paradigma pemrograman ini merupakan trend baru dan sangat populer akhir-akhir ini. Paradigma pemrograman yang lain adalah pemrograman fungsional, pemrogramn deklaratif, dan pemrograman konkuren.

5.4 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, tetapi 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 T0 dan berakhir pada saat T1. Maka efek neto dari aksi dapat dijelaskan dengan membandingkan keadaan pada saat T0 dan keadaan pada saat T1.. 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), T0, 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), T1, kentang dalan keadaan terkupas di panci, siap untuk dimasak dan kantong kentangnya harus dikembalikan ke rak lagi.

Pengandaian yang lain adalah bahwa persediaan kentang 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 makan 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 dititik beratkan efek netonya, yaitu keadaan " sebelum dan sesudah" maka kejadian tersebut dianggap sebagai berarti kejadian tersebut dianggap sebagai proses. Dengan anggapan kejadian tersebut suatu proses, maka T0 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 T1 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 uruturutan 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 T0). 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 kejadian 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 memprogam.

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 bahwa 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 sepakati 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 subalgoritma, 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 Tati yang mengupas kentang :

a. Ibu Tati mengambil kantong kentang dari rak.
b. Ibu Tati mengambil panci dari almari.
c. Ibu Tati mengupas kentang.
d. 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 :

a. Ambil kantong kentang dari rak
b. Ambil panci dari almari
c. Kupas kentang
d. Kembalikan kantong kentang ke rak

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

a. Ina mengambil kantong kentang dari rak
b. Ina mengambil panci dari almari
c. Ina mengupas kentang
d. 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 :

a. Aida mengambil kantong kentang dari rak.
b. Aida mengambil panci dari almari.
c. Aida mengupas kentang.
d. 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 dari 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 :

a. Ibu Tati mengambil kantong kentang dari rak.
b. Ibu Tati mengambil panci dari almari.
c. Ibu Tati memakai celemek.
d. Ibu Tati mengupas kentang.
e. Ibu Tati mengembalikan kantong kentang ke rak.

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

a. Ibu Tati mengambil kantong kentang dari rak.
b. Ibu Tati mengambil panci dari almari.
c. Ibu Tati mengupas kentang.
d. Ibu Tati mengembalikan kantong kentang dari rak.

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

a. Ambil kantong kentang dari rak.
b. Ambil panci dari almari.
c. Lakukan persiapan, tergantung pakaian.
d. Kupas kentang.
e. 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 dapat 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*).

a. Ambil kantong kentang dari rak
b. Ambil panci dari almari
c. if baju berwarna muda then Pakai celemek
d. Kupas kentang
e. 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 :

a. Ambil kantong kentang dari rak
b. Ambil panci dari almari

c. ***if*** baju berwarna muda then pakai celemek
d. ***while*** jumlah kentang terkupas belum cukup ***do*** Kupas 1 kentang
e. 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 :

a. Ambil kantong kentang dari rak
b. Ambil panci dari almari
c. if baju berwarna muda then pakai celemek
d. while jumlah kentang terkupas belum cukup do Kupas 1 kentang
e. 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 :

a. Ambil kantong kentang dari rak
b. Ambil panci dari almari
c. depend on baju berwarna muda : pakai celemek tidak berwarna muda :-
d. while jumlah kentang terkupas belum cukup do
e. Depend on kantong kentang
o ada isinya : Kupas 1 kentang
o 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 standar 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 pengalamn, 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 primitive 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

[1]. 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 hypotesa : 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

[2]. Distribusi gula-gula

1. 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 keempat anak tersebut, maka sisanya tidak dibagikan. Tuliskanlah algoritmanya.
2. 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.

BAB VI
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

Dalam sebuah teks algoritmik adalah **sesuatu** yang dipakai sebagai identifikasi :

- modul **program**, **algoritma**, **skema** program dsb.
- **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, maka 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 symbol 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. Nama karena merupakan satu kesatuan ***leksikal***, maka 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 dimana 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 procedural dapat berupa:

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

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 tsb.
- konvensi atau perjanjian tentang penulisan **konstanta** bertype tsb.
- **operator** yang dapat dioperasikan terhadap objek bertype tsb.

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, maka 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.

1. 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 table di atas.

2. 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 predesesor -1 adalah -2
suksesor -1 adalah 0 predesesor 3 adalah 2
suksesor 5 adalah 6

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

KELOMPOK	Op	ARTI	Hasil
	^	Pangkat	integer
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

3. 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

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 * 102
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.

4. 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 (maka 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, maka 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 nama type < elemen1: type1,
elemen2: type2,
elemen2: type2,
…….>

Perhatikan dalam pengertian sebagai type bentukan, maka ada kesleuruhan type yang harus dibentuk menurut pembentuk tertentu yaitu **Konstruktor** pada perkuliahan terkait), 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 Point.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 Date : **< 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] dan SS bernilai [0..59] dan ampm yang merupakan enumerasi (am, pm) }
type Date : < 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**
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 : < Dosen,Matakuliah>
Type terstruktur Kuliah :
type Kuliah <Dari,Ke:jam,DMK: DosenMK, PS: ProgramStudi >

Cara mengacu nilai yang tersimpan pada
Kuliah_SKI yang bertype Kuliah adalah:
Kuliah_SKI.Dari {< integer[11..610]>}
Kuliah_SKI.Ke { < integer [11..610]>}
Kuliah_SKI.DMK { DosenMK}
Kuliah_SKI.DMK.D { string}
Kuliah_SKI.DMK.MK { string}
Kuliah_SKI.PS {string}

Domain : sesuai dengan domain masing-masing komponen
Konstanta : < 15,16,<"Marni", 9012>, "IF"> < 21,22,<"Edi", 9012>, "TF">

Nilai, Ekspresi, Input & 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:

1. *assignment*, atau
2. dibaca dari suatu piranti masukan

Tugas latihan

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.

Memprogram 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
   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 dituliskan
                         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 :

- 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*, dsb). Disebut "dibaca", karena arah dari pengisian harga yaitu seakan-akan computer "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 dihhubungkan 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.

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 ybs.), 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 variable

```
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 variable.

```
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 }
```

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 variable.

```
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"	

BAB VII
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 k-2, ke-3,.... s/d ke-n. Program paling sederhana tentunya hanya mengandung 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 sekuaential

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 }
```

Contoh aksi sekuensial yang tidak berpengaruh jika diubah urutannya.

```
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. Dengan menuliskan semacam ini, program akan dapat dioperasikan dengan lebih mudah, namun algoritma menjadi sangat rinci. Lebih rinci lagi, dapat

dibuat *layer* 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 : JAM MENIT DETIK

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.

Contoh5 : 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..24],
                 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 computer 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 perjam 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 ?.

BAB VIII
ANALISIS KASUS

Analisis kasus, yang melahirkan instruksi kondisional, adalah elemen primitif pembangun algoritma, yaitu memungkinkan kita untuk membuat teks yang sama, tetapi 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* ybs. 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 **:**

1. *kondisi*-1,*kondisi*-2,*kondisi*-3 ... .,*kondisi*-N domain harganya [**true**,**false**]
2. *kondisi*-1,*kondisi*-2,*kondisi*-3 ... .,*kondisi*-N adalah ekspresi lojik/boolean yang mengandung nama-nama sebagai operan.
3. *kondisi*-1 ∩ *kondisi*-2 ∩ *kondisi*-3 ... . ∩ *kondisi*-N = Ø. Berarti semua kondisi disjoint, tidak ada kasus yang sama tercakup pada dua buah kondisi.
4. *kondisi*-1 ∪ *kondisi*-2 ∪ *kondisi*-3 ∪ *kondisi*-N 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. Di sini, "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 b>a. Tidak mungkin keduanya.

Spesifikasi:

Input : a dan b integer

Proses : menuliskan harga yang lebih besar, dengan spesifikasi bahwa nilai a yang akan ditulis, jika a ≤ b.

Output : a atau b, integer

```
Program MAXAB
{dibaca nilai a dan b,menuliskan a jika a≤ b,menuliskan b jika b > a}

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 oC) 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 ≤ 0
"Cair" jika 0 < T ≤ 100
"Uap" jika T > 100

```
Program WUJUDAIR1
{Dibaca  T(integer), temperatur air (dalam OC) 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 OC) 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 dan 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, 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 ≠ b, b ≠ c, a ≠ 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 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
                         yg menyatakan validitas dari data }
     { Lakukan proses }


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

Contoh predikat yang merupakan ekspresi data valid:

5. jika data harus positif, dan data adalah X bertype integer : $X \geq 0$
6. 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:

AX2 + 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,-. Pegawai non-staf yang telah bekerja lebih dari 5 tahun akan mendapat bonus sebesar Rp. 400.000,- jika berumur lebih dari 50 tahun, sedangkan pegawai nonstaf yang berumur kurang dari 50 tahun hanya akan mendapat bonus Rp. 250.000,-. Pegawai staf yang umurnya kurang dari 50 tahun akan mendapat bonus Rp. 300.000,-

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 bertipe 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,- Rp. 5000,-
2 Rp. 300.000,- Rp. 3000,-
3 Rp. 250.000,- Rp. 2000,-
4 Rp. 100.000,- Rp. 1500,-
5 Rp. 50.000,- Rp. 1000,-

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.

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")
```

8. Diberikan deretan analisis kasus sebagai berikut.
Berikan komentar mengenai analisis kasus yang dituliskan dalam teks algoritma sebagai berikut. Suatu ruangan mempunyai tiga 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.

Contoh Fungsi:

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

f(x) = x2 + 3x – 5

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

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

f(x,y) = x2 + 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

1. Pendefinisian/Spesifikasi fungsi

1.

```
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, fungsi tidak membutuhkan apa-apa dari pemakainya untuk menghasilkan harga.
- jika *list* parameter *input* (parameter formal) tidak

kosong, minimal mengandung satu nama, maka nama tersebut harus berupa nama informasi beserta typenya.

- 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). Tipe hasil boleh dinyatakan oleh suatu nama tipe. Dengan catatan, bahwa instruksi "terakhir" belum tentu dituliskan pada baris terakhir, misalnya jika hasil merupakan sebuah nilai yang dikirimkan berdasarkan analisis kasus.

```
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 )
```

2. Pemanggilan Fungsi

Catatan :

1. Pada waktu pemanggilan terjadilah asosiasi antara parameter formal/input dengan parameter aktual sesuai dengan urutan penulisan dalam list-nama parameter input.
2. *List* 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 typenya dengan list parameter formal 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.

Fungsi boleh dipakai oleh program utama, prosedur atau fungsi lain.

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))
```

Contoh1-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-Fungsi : 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
{Mentest 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
```

Contoh3-Fungsi : 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 funsgi di atas ke dalam bahasa Pascal, C dan Ada. Akan terlihat perbedaan yang sangat besar di antara ketiganya.

Contoh 4-Fungsi : Ubah dan Periksa
KarPersoalan :

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 Karakter ToInteger 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 contoh1

Catatan :

Test yang dilakukan sebelum pemanggilan fungsi Karakter ToInteger membuat fungsi Karakter

ToInteger 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 di luar 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-Fungsi : HITUNG FUNGSI
Definisi dan pemanggilan fungsi
Persoalan

Tuliskanlah algoritma yang membaca 3 bilangan bulat (a,b,c), dan menghitung:
6*(ax2 + bx +c) untuk x = 1

Spesifikasi

Input : a,b,c bilangan bulat, koefisien persamaan kuadrat, x = 1
Output : Fx

Proses : menghitung Fx = 6*(ax2 + bx +c) menuliskan hasil

```
Program HITUNGFUNGSI
{ Program membaca tiga buah bilangan bulat a,b,c dan menhitung 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)
```

```
funotion  FungsiKUADRAT  (a,b,c,x : integer) →  integer
{Diberikan a,b,c ,  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 :

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

Contoh-7: PENANGGALAN & NEXTDAY

Pemetaan type bentukan ke type bentukan

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

type Nama : DATE <Tanggal, Bulan,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 dr 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.D+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.Tanggal :
            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:

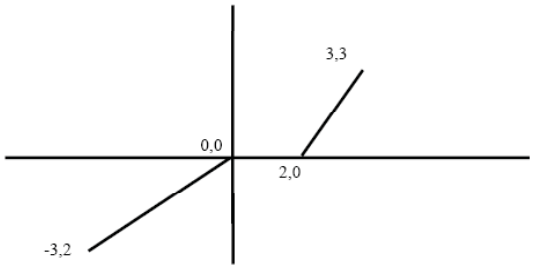
- Tuliskanlah definisi fungsi KABISAT dan algoritmanya.
- Buat analisis kasus yang lain dari yang dituliskan pada fungsi NEXTDAY di atas, dengan mulai menganalisis tanggal dan bukan bulan.
- Buatlah kode program dari fungsi Nextday yang sama dengan hanya melakukan komputasi aritmatika, seperti halnya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

Latihan Soal Fungsi

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 bertype dasar. Bahkan pada FORTRAN hanya dapat menghasilkan type numeric (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 :



Pecahan

1. Definisikan sebuah tipe 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 atas 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 }
```

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 }
```

BAB IX
PROSEDUR

9.1 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 tipenya.

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.

9.2 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.

9.3 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, tipe harus kompatibel.

Beberapa bahasa pemrograman, dapat diasosiasikan 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 independen.

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.

9.4 Notasi Algoritmik untuk Prosedur

Pada bagian ini diberikan skema pendefinisian dan pemanggilan prosedur.

Pendefinisian/Spesifikasi Prosedur

```
procedure NAMAPROSEDUR ( [list nama parameter formal: type])
{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 tipenya.

Pemanggilan Prosedur

```
Program POKOKPERSOALAN
{Spesifikasi , Input, Proses, Output}
Kamus :
{ semua NAMA yang dipakai dalam algoritma }

   procedure NAMAPROSEDUR  (Input/Output : <list-nama 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*-nama parameter

formal.

- *List* parameter aktual harus sama jumlah, urutan dan tipenya 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-Prosedur: 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 ohm}
   V : integer    { tegangan dalam ohm}
   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}
{ FS  : 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 ohm}
   V : integer    { tegangan dalam ohm}

   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}
{ FS  : memproses R dan A sehingga dihasilkan V yaitu tegangan dengan
rumus : V = R * A}
Kamus lokal :
Algoritma :
   V ← R * A
```

Catatan :

- Prosedur dengan parameter lebih menjamin modularitas program. Sedapat mungkin semua prosedur diparametrisasi dengan baik.
- 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.
- 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,  }
{ FS : 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 }
```

Contoh3-Prosedur : PUTAR3BIL

Contoh pemanfaatn 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?

BAB X
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 atas dua bagian :

- Kondisi yang mengakibatkan pengulangan suatu saat berhenti, yang dinyatakan oleh sebuah ekspresi logik baik secara eksplisit maupun implicit;
- 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 hakikatnya notasi pengulangan hanyalah merupakan sebagian dari skema pengulangan yang akan dibahas pada bab-bab berikutnya.

1. Macam notasi pengulangan:

1 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.

1 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*. Test 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*, test *kondisi-berhenti*, [*Aksi*, test *kondisi-berhenti*.. dst] diakhiri test- *kondisi-berhenti* yang bernilai true, yang menyebabkan pengulangan berhenti.

1 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 *test* terhadap kondisi berhenti. *Test* terhadap *kondisi-pengulangan* dilakukan setiap kali sebelum *Aksi* dilaksanakan. Pengulangan ini berpotensi untuk menimbulkan aksi "kosong" (tidak pernah melakukan apa-apa, karena pada *test* yang pertama, *kondisi-pengulangan* tidak dipenuhi (bernilai *false*).

Aksi sekuensial yang dilakukan adalah : test *kondisi-pengulangan*, [*Aksi*, test *kondisipengulangan*.. dst] diakhiri test- *kondisi-pengulangan* yang bernilai false, yang menyebabkan pengulangan berhenti.

1 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 test terhadap kondisi berhenti. Tergantung kepada kondisi berhenti yang ditest:

- *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,* test *kondisi-berhenti*, [*Aksi-2*, test *kondisi-berhenti, ..*] diakhiri test- *kondisi-berhenti* yang bernilai true, yang menyebabkan pengulangan berhenti.

1 Pengulangan Berdasarkan Pencacah

nama-pencacah **traversal** [*range harga*]
Aksi
{Catatan : *nama--pencacah* harus suatu tipe 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 tipe 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 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 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 *output*-nya 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:

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 TULISBIL1
{Dibaca N ≥ 0,  Menuliskan 1,2,3... N berderet ke bawah, dengan
bentuk repeat..N times }
Kamus:
   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:
   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:
   i : integer {bilangan yang akan ditulis }
Algoritma :
   input (N)
   i ← 1
   while i ≤ N do
      Output (i)
      i ← i + 1
   { i > N )
```

Penjelasan :

- Dengan bentuk ini, nilai 1 belum tentu ditulis. Jika ternyata pengguna memberikan nilai N yang negatif, program tidak menuliskan apa-apa karena kondisi yang diperiksa pertama kali (i ≤ N) menghasilkan false.
- Nama yang didefinisikan sebagai i akan bernilai 1..N+1.
- 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:
   i : integer     { nilai yang akan ditulis }

Algoritma :
   input (N)
   i  ← 1
   iterate
      Output (i)
   stop  (i = N)
      i   ←  i + 1
   { i = N}
```

Penjelasan:

- Dengan bentuk ini, nilai 1 pasti ditulis. Jika ternyata pengguna memberikan nilai N yang negatif, program akan menuliskan 1 kemudian berhenti.
- Nama yang didefinisikan sebagai i akan bernilai 1..N , jika N positif
- 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 iterate }
Kamus:
   i : integer     { nilai yang akan ditulis }

Algoritma :
   input (N)
   i traversal [1..N]
      Output (i)
```

Penjelasan:

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

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 penjelasan mengenai algoritma dan pemrograman ini, disebutkan bahwa 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.

BAB XI
SKEMA PEMROSESAN SEKUENSIAL

11.1 Pemrosesan Sekuensial

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 bertipe dasar (*integer, real, character, boolean*), tetapi dapat juga bertipe 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, dsb. 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 pada bab sebelumnya hanya berfokus

kepada *bentuk pengulangan*, belum mencakup semua abstraksi yang dibutuhkan untuk suatu proses sekuensial.

11.2 Spesifikasi Primitif

Primitif untuk skema sekuensial diterjemahkan menjadi prosedur dan nilai variable 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_Elemt dari Current_ Elmt adalah Next_Elmt + 1; dengan syarat First_Elmt diketahui yaitu=1.

2. Pemrosesan data bertype bentukan :

MHSNILAI : tipe <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 satu pun mahasiswa 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 bertype **MHSNILAI**

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

Next_Elmt akan mengakibatkan harga Current_Elmt yang baru = <8689007,"IF221", 'B'>

jika Current_Elmt = <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 thd 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.

```
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: akan diproses}
   iterate
      Output (i)
      Sum  ← Sum + i
   stop( i=N )   {EOP}
      i   ←  i + 1             {Next_Elmt}
   Output (Sum)               {Terminasi}
```

Versi-1

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 skuensial 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 :

- Pengontrol pengulangan adalah bilangan integer i
- 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
- Program benar, dengan tambahan spesifikasi : N ≥ 1, sesuai dengan definisi domain i = 1..N
- Nilai i di luar definisi domain 1..N
- 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.
2. Program benar, dengan tambahan spesifikasi : N ≥ 1, sesuai dengan definisi domain i = 1..N
3. 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 utk dijumlahkan,
                    pembacaan diakhiri dg 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 +   ....    s+   ... Xi-1 . }
   Output (Sum) {Terminasi}
```

Penjelasan :

1. Pengontrol pengulangan (*Current element*) adalah nilai X.
2. Program ini *memakai skema yang benar*, tetapi tidak menangani kasus kosong.
3. Program tersebut benar untuk kasus kosong atau pun 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.

BAB XII
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*, tetapi tidak menangani kasus kosong.
3. Program tersebut benar untuk kasus kosong atau pun 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*, tetapi tidak menangani kasus kosong.
3. Program tersebut benar untuk kasus kosong atau pun 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, tetapi 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*, tetapi 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*)

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 dg 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 di baca,
      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*, tetapi tidak menangani kasus kosong.

3. Program tersebut benar untuk kasus kosong atau pun 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*, tetapi *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 dg 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 di baca,
      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*, tetapi 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 dg 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 di baca,
         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 dg 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 di baca,
         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 dg 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 di baca,
      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.

BAB XIII
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:

Si = (-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)
```

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

BAB XIV
DASAR PEMROGRAMAN GRAFIK

Gambar bermakna sejuta kata. Kalimat tersebut tidak berlebihan, karena dengan gambar kita dapat memperlihatkan sesuatu tanpa perlu menjelaskan, dan orang pun langsung mengerti. Pada bagian ini, kita akan membahas mengenai dasar-dasar pemrograman grafik menggunakan bahasa C.

Ada dua syarat utama untuk memulai pemrograman grafik, yaitu meng-*include*-kan file header <graphics. h> dan melakukan proses inisialisasi mode grafik yang digunakan oleh komputer. Untuk inisialisasi mode grafik digunakan perintah initgraph() yang ada di file header graphics.h. Format umum dari perintah ini adalah : initgraph(graphdriver, graphmode, direktori_bgi)

Variabel graphdriver digunakan untuk menyimpan nilai driver grafis yang ditentukan berdasarkan kartu grafik yang digunakan. Variabel graphmode digunakan untuk menyimpan nilai mode grafis, sedangkan direktori_ bgi adalah tempat file .bgi yang digunakan dalam pemrograman grafik. Contoh listing program berikut ini akan menggambarkan persegi panjang dengan diagonal yang menutupi layar.

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
```

```
int Xmax, Ymax;
int GraphDriver,GraphMode,ErrorCode;
main()
{
        GraphDriver = DETECT;
        initgraph( &GraphDriver, &GraphMode, "c:\\tc" );
        ErrorCode = graphresult();
        if( ErrorCode != grOk ){
              printf(" Graphics System Error: %s\n",
              grapherrormsg( ErrorCode ) );
              exit( 1 );
              }
        Xmax = getmaxx();
        Ymax = getmaxy();
        rectangle(0,0,Xmax,Ymax);
        line(0,0,Xmax,Ymax);
        line(0,Ymax,Xmax,0);
        getch();
        closegraph();
        return 0;
 }
```

Program di atas dimulai dengan inisialisasi variabel GraphDriver = DETECT. Perintah ini berarti program meminta *autodeteksi*, sehingga Turbo C secara langsung akan mencocokkan *driver* dan mode grafik yang terdeteksi. Proses berikutnya adalah inisialisasi mode grafik dengan perintah :

```
initgraph( &GraphDriver, &GraphMode, "c:\\tc" );
```

melalui perintah ini Turbo C akan menginisialisasi kartu grafik yang telah terdeteksi, kemudian kompiler akan menemukan dan memasukkan *graphic-driver* (file dengan akhiran .bgi). Begitu nama driver ditemukan, maka driver

akan menangani rincian antara program grafik dengan perangkat keras.

Jika ternyata driver grafik tidak dapat ditemukan, maka initgraph tidak akan dapat membuat grafik apa pun. Karena itu, perintah ErrorCode = graphresult(); digunakan untuk mengembalikan nilai inisialisasi grafik. Jika nilai yang dikembalikan 0 (grOK) maka kita dapat memulai program grafik. Jika tidak, maka kita harus memuat listing berikut untuk mendeteksi kesalahan pada inisialisasi grafik.

```
if( ErrorCode != grOk ){
        printf(" Graphics System Error: %s\n",
        grapherrormsg( ErrorCode ) );
        exit( 1 );
}
```

Setelah proses inisialisasi berhasil dilakukan, maka kita dapat mulai membuat grafik yang diinginkan. Pixel pada layar dinomori mulai dari sudut kiri atas layar (0, 0) sampai dengan sudut kanan bawah (getmaxx(), getmaxy).

Perintah untuk menggambar persegi panjang adalah rectangle(x1,y1,x2,y2); dimana x1 dan y1 adalah koordinat sudut kiri atas dari persegi panjang, dan x2, y2 adalah sudut kanan bawah dari persegi panjang. Sedangkan perintah line(x1,y1,x2,y2); digunakan untuk menggambar garis dari koordinat (x1,y1) sampai koordinat (x2,y2). Jika

program selesai dilakukan maka kita harus memanggil fungsi closegraph(); untuk menutup mode grafik.

Fungsi-fungsi Gambar Pada Turbo C

Dengan menggunakan format program seperti contoh sebelumnya, kita dapat membuat gambar-gambar lain menggunakan fungsi-fungsi gambar yang disediakan oleh turbo C, diantaranya adalah :

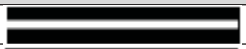
1. Garis

Untuk menggambar garis digunakan fungsi line(x1,y1,x2,y2) yang berarti penggambaran garis dimulai dari koordinat awal (x1, y1) sampai koordinat akhir (x2,y2). Perintah lain untuk menggambar garis adalah lineto(x,y) dan linerel(dx,dy). Untuk menggunakan kedua perintah tersebut, kita harus mengerti terlebih dahulu konsep CP (*Current Position*) yang berarti posisi saat ini. Pada saat pertama kali memanggil fungsi lineto(x,y) dan linerel(dx,dy)CP terletak pada posisi koordinat (0,0) sehingga garis akan digambar dari posisi (0,0) sampai posisi (x,y) pada perintah lineto() dan posisi (dx,dy) pada perintah linerel(), selanjutnya CP akan diperbaharui pada titik akhir penggambaran garis.

Pada awal pemanggilan kedua perintah tersebut memang memiliki persamaan, yaitu menggambar garis dari posisi (0,0) sampai ke posisi pada parameter perintah.

Perbedaanya adalah saat CP diperbaharui pada posisi akhir penggambaran garis (xCP,yCP). Jika kita menggunakan fungsi lineto(x,y)maka garis akan digambar dari (xCP,yCP) sampai koordinat (x,y), kemudian CP akan diperbaharui pada posisi (x,y). Pada penggunaan perintah linerel(dx,dy), garis akan digambarkan dari koordinat

(xCP,yCP) sampai koordinat (xCP+dx , yCP+dy). Turbo C juga menyediakan bentuk-bentuk garis yang dapat diatur dengan perintah setlinestyle(tipe_garis, pola_garis, tebal_garis). Nilai untuk tipe garis diberikan dalam tabel berikut ini :

NAMA GARIS	NOMOR GARIS	GAMBAR
SOLID_LINE	0	
DOTTED_LINE	1	
CENTER_LINE	2	
DASHED_LINE	3	

Nilai untuk pola garis diabaikan sehingga kita dapat mengisikan nilai nol (0). Sedangkan ketebalan garis memiliki pilihan nilai sebagai berikut :

KETEBALAN	NILAI	KETERANGAN
NORM_WIDTH	1	Ketebalan 1 pixel
THICK_WIDTH	3	Ketebalan 3 pixel

Penggunaan perintah ini dapat dilakukan dengan langsung mengisikan tipe garis, atau dengan nomornya saja. Jadi perintah :

Setlinestyle(center_line,0,thick_width) sama dengan
Setlinestyle(2,0,3)

2. Segi Empat

Untuk menggambar segi empat, digunakan perintah rectangle(x1,y1,x2,y2), dimana x1 dan y1 adalah koordinat sudut kiri atas dari persegi panjang, dan x2, y2 adalah sudut kanan bawah dari persegi panjang. Perintah lain untuk menggambar segiempat adalah bar(x1,y1,x2,y2). Perbedaan kedua perintah ini terletak pada pewarnaan (akan dijelaskan pada bagian warna grafik).

3. Lingkaran

Untuk menggambar lingkaran digunakan perintah circle(x,y,radius) di mana (x,y) adalah koordinat titik pusat lingkaran, sedangkan radius adalah jari-jari lingkaran.

4. Elips

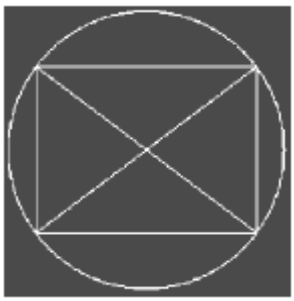
Untuk menggambar elips digunakan perintah ellips e(x,y,alpha,beta,rx,ry) dimana (x,y) adalah koordinat titik pusat elips, alpha adalah sudut awal ellipse, beta adalah sudut akhir ellipse, rx adalah jari-jari ellipse yang sejajar dengan sumbu x, ry adalah jari-jari ellipse yang sejajar dengan sumbu y.

Ellips akan digambarkan tertutup atau terbuka sesuai dengan nilai alpha dan betanya, jika ellips ingin digambar tertutup, maka alpha harus diisi dengan 0, dan beta harus diisi dengan 360. perintah ellipse() juga dapat digunakan untuk menggambar lingkaran yaitu dengan mengisi nilai alpha = 0, beta = 360 dan rx = ry. Jika ingin menggambarkan *ellipse* dengan pola isian warna, digunakan perintah fillellipse(x,y,rx,ry) dimana (x,y) adalah koordinat titik pusat ellipse, rx jari-jari terpanjang ellipse, dan ry jari-jari terpendek *ellipse*. Parameter *alpha* dan *beta* memang tidak terdapat pada perintah ini, karena *ellipse* akan diisi pola isian warna maka *ellipse* harus digambar tertutup dengan kata lain nilai alpha selalu 0 dan beta selalu 360 sehingga tidak perlu dituliskan. Pola isian warna akan dibahas berikutnya. Contoh program menggunakan fungsi gambar diberikan pada listing berikut ini :

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
main()
{
```

```
        int x,y;
        int GraphDriver,GraphMode,ErrorCode;
        GraphDriver = DETECT;
        initgraph( &GraphDriver, &GraphMode, "c:\\tc" );
        ErrorCode = graphresult();
        if( ErrorCode != grOk ){
              printf(" Graphics System Error: %s\n",
              grapherrormsg( ErrorCode ) );
              exit( 1 );
              }
        x=getmaxx();
        y=getmaxy();
        rectangle(x/4,y/4,3*x/4,3*y/4);
        line(x/4,y/4,3*x/4,3*y/4);
        line(3*x/4,y/4,x/4,3*y/4);
        circle(x/2,y/2,200);
        getch();
        closegraph();
        return 0;
}
```

Hasil eksekusi program di atas ditunjukkan oleh gambar berikut :



Warna Dan Pola Pada Grafik

Warna grafik yang disediakan turbo C pada file header graphics.h adalah :

KODE WARNA	KETERANGAN
0	BLACK
1	BLUE
2	GREEN
3	CYAN
4	RED
5	MAGENTA
6	BROWN
7	LIGHTGRAY
8	DARKGRAY
9	LIGHTBLUE
10	LIGHTGREEN
11	LIGHTCYAN
12	LIGHTRED
13	LIGHTMAGENTA
14	YELLOW
15	WHITE

Perintah-perintah yang berhubungan dengan pola dan warna di antaranya :

1. setcolor()

Perintah setcolor() berfungsi untuk menentukan warna gambar yang aktif. Sintaks dari perintah ini adalah setcolor(nomor_warna). Contoh, jika kita menuliskan perintah:

```
setcolor(1);
rectangle(50,50,120,100);
```

berarti kita menggambarkan segi empat dengan warna border biru.

2. setbkcolor()

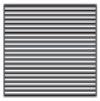
Perintah setbkcolor() berfungsi untuk mengatur warna background layar. Sintaks dari perintah ini adalah setbkcolor(nomor_warna). Contoh, untuk membuat *background* berwarna kuning digunakan perintah setbkcolor(14).

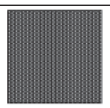
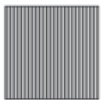
3. floodfill()

Sintaks dari perintah ini adalah floodfill(x, y, warna_ border). Perintah floodfill() berfungsi untuk mengisi suatu wilayah dari posisi (x,y) sampai warna border yang sudah ditentukan. Contohnya, perintah floodfill(100,50,4)digun akan untuk mengisi warna dari posisi koordinat (100,50) sampai ditemukan warna border merah.

4. setfillstyle()

Perintah setfillstyle() berfungsi untuk msengatur warna dan pola isian. Sintaks dari perintah ini adalah setfillstyle(nomor_pola, nomor_warna). Jenis-jenis pola isian diberikan oleh tabel berikut :

NAMA POLA	NOMOR POLA	GAMBAR
EMPTY_FILL	0	
SOLID_FILL	1	
LINE_FILL	2	
LTSLASH_FILL	3	

SLASH_FILL	4	
BKSLASH_FILL	5	
LTBKSLASH_FILL	6	
HATCH_FILL	7	
XHATCH_FILL	8	
INTERLEAVE_FILL	9	
WIDE_DOT_FILL	10	
CLOSE_DOT_FILL	11	

Teks dalam Mode Grafik

Untuk menampilkan teks dalam mode grafis digunakan perintah outextxy(x,y,str), dimana (x,y) adalah koordinat awal penulisan teks dan str adalah teks yang akan dituliskan. Fungsi outtextxy akan selalu menuliskan teks berdasarkan koordinat tetapi tidak memperbaharui CP (*Current Position*).

Fungsi lain untuk menuliskan teks adalah perintah outtext(str). Fungsi outtext() akan menuliskan teks

dari posisi(0,0) saat pemanggilan pertama, kemudian memperbaharui CP pada pemanggilan berikutnya berdasarkan posisi CP sebelumnya. Untuk mengatur jenis dan ukuran font, digunakan perintah settextstyle().

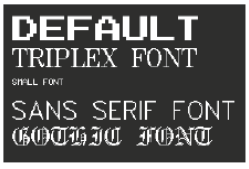
Sintaks dari perintah ini adalah: Jenis-jenis font yang dapat digunakan dalam bahasa C diberikan dalam tabel berikut ini :

NAMA FONT	NILAI	KETERANGAN
DEFAULT_FONT	0	Font 8x8 bit-map
TRIPLEX_FONT	1	Font triplex
SMALL_FONT	2	Small font
SANS_SERIF_FONT	3	Font sans serif
GOTHIC_FONT	4	Font gothic

Arah penulisan tabel terdiri atas dua jenis, yaitu dari kiri ke kanan atau dari atas ke bawah. Parameter arah diberikan dalam tabel berikut ini :

DIRECTION	NILAI	KETERANGAN
HORIZ_DIR	0	Dari kiri ke kanan
VERT_DIR	1	Dari atas ke bawah

Sedangkan ukuran font ditentukan dari nilai 1 sampai 10. Angka 1 menggambarkan teks dengan ukuran huruf 8x8 pixel, 2 adalah 16x16 pixel demikian seterusnya. Contoh bentuk dari jenis-jenis font di atas dapat dilihat berikut ini : settextstyle(jenis, arah, ukuran)



Berikut ini akan diberikan contoh listing program yang menampilkan teks dan grafik :

```
#include<stdio.h>
#include<conio.h>
```

```
#include<graphics.h>
int x,y;
int GraphDriver,GraphMode,ErrorCode;
main()
{
        GraphDriver = DETECT;
        initgraph( &GraphDriver, &GraphMode, ”c:\\
        tc” );
        ErrorCode = graphresult();
        if( ErrorCode != grOk ){
        printf(” Graphics System Error: %s\n”,
        grapherrormsg( ErrorCode ) );
        exit( 1 );
}
settextstyle(3,0,4);
outtextxy(25,50,”HAI FRIEND,”);
outtextxy(25, 100,”NAMA SAYA MR. SMILE");
x = getmaxx()/2;
y = getmaxy()/2;
bar(x-75,y-50,x+75,y+100);
setcolor(0);
setfillstyle(9,14);
fillellipse(x,y+25,60,60);
setfillstyle(10,0);
fillellipse(x-25,y+10,5,5);
fillellipse(x+25,y+10,5,5);
setcolor(0);
setlinestyle(0,0,3);
ellipse(x,y+40,180,360,20,10);
getch();
closegraph();
return 0;
}
```

BAB XV
PEMROGRAMAN GRAFIK

Pemrograman Grafik adalah suatu tahap pemrograman yang membutuhkan pengetahuan tentang teknik pemrograman ditambah dengan penguasaan yang cukup tentang tata koordinat (yaitu koordinat Cartesian, yang menggunakan sumbu X dan Y). Penyajian hasil pengolahan dalam wujud grafik memberikan informasi yang jauh lebih efektif daripada informasi yang hanya dalam bentuk teks.

15.1 Penyiapan Pemrograman Grafik

Statemen SCREEN

Fungsi : statemen untuk melakukan perpindahan / penyiapan ke mode grafik

Bentuk umum :

SCREEN [mode] [,[kode] [,[ha][,hv]]

Penjelasan :

mode bila diisi :

0 = mode biasa, resolusi layar dalam 40 x 80

1 = grafik resolusi medium layar dibagi dalam 320 x 200

2 = grafik resolusi tinggi layar dibagi dalam 640 x 200

kode : berisi bilangan untuk pengatur warna.

Bila mode 0 maka kode bernilai 0 warna akan hitam putih saja. Bilangan lain memungkinkan pemberian

warna. Bila resolusi medium maka akan sebaliknya, yaitu bila kode berisi 0 maka akan memungkinkan pemberian warna.

ha : halaman aktif, dapat dipakai hanya bila modenya 0.

hv : halaman visual, untuk memilih halaman yang akan

Screen no	Screen Mode	Kemampuan
0	Mode teks	Teks, 16 warna
1	Mode grafik resolusi menengah	Teks,Grafik 8 warna (2x4 warna)
2	Mode grafik resolusi tinggi	Teks, Grafik, hitam-putih

Program

Statemen COLOR

Fungsi : mengatur warna untuk statemen dasar grafik yang lain, yaitu : untuk statemen PSET, PRESET, LINE, CIRCLE, PAINT dan DRAW.

Bentuk umum :

COLOR [b][,p]

Penjelasan :

b : warna latar belakang, pilihan warna 0-15

p : warna garis yang digambar, dipilih warna 0-15

15. 2 Bentuk Dasar Pemrograman Grafik

Statemen PSET

Fungsi : untuk menggambar sebuah titik pada layar.

Bentuk umum :

PSET (x,y) [,warna]

Penjelasan :

x : absis x pada layar

y : ordinat y pada layar

warna : bilangan kode warna , dapat berisi 0-3
Bila tak disebutkan, maka harganya dianggap 3 untuk resolusi medium dan 1 untuk resolusi tinggi.

Harga x dan y dapat berharga di luar kapasitas layar (320, 200) untuk resolusi medium dan (640, 200) untuk resolusi tinggi, hanya saja tak kelihatan di layar.

Statemen PRESET
Fungsi : untuk menggambar titik pada layar

Fungsinya sama dengan PSET, perbedaannya terletak pada nilai yang otomatis akan diberi bila parameter warna tak diisi. Statemen PRESET dengan warna 0, biasanya dipakai untuk menghapus gambar hasil PSET.
Bentuk umum :
PRESET (x,y),warna
Penjelasan :
x : absis x pada layar
y : ordinat y pada layar
warna : warna gambar, bila tak disebut maka diberi nilai 0 yang berarti sama dengan latar belakangnya, dengan kata lain tak tampak gambar apa-apa.

Statemen LINE
Fungsi : untuk menggambar garis pada layar.
Bentuk umum :
LINE [(x1,y1)] – (x2,y2) [,[warna][,B[F]][,style]]

Penjelasan :
x1,y1 : koordinat awal garis yang akan ditarik. Dapat koordinat absolut maupun relatif. Bila dipakai untuk gambar kotak maka ini adalah koordinat kiri atas kotak.

x2,y2 : koordinat akhir titik garis yang ditarik. Dapat koordinat absolut maupun relatif. Bila dipakai untuk menggambar kotak maka ini adalah koordinat pojok kanan bawah.
warna : warna untuk menggambar. Dapat berisi 0-3
B : berarti menggambar kotak
F : kotak yang digambar dicat.
style : berisi bilangan 0 - hFFFF untuk "mask" penggambaran di layar.

Statemen DRAW

Fungsi : untuk menggambar suatu bentuk tertentu seperti yang disebutkan dalam ekspresi.
Bentuk umum :
DRAW ekspresi
Penjelasan :
Ekspresi :

Penggambaran diawali dari titik terakhir yang digambar sebelum statemen ini. penggambaran dapat dilakukan dengan perintah-perintah sebagai berikut :

- U [n] : ke atas [n skala]
- D [n] : ke bawah
- L [n] : ke kiri
- R [n] : ke kanan
- E [n] : diagonal ke atas dan ke kanan
- F [n] : diagonal ke bawah dan ke kanan
- G [n] : diagonal ke bawah dan ke kiri
- H [n] : diagonal ke atas dan ke kiri

Bila skala "n" tak ditulis, maka dianggap 1.

M x,y : dari titik terakhir bergerak ke koordinat x,y dan ditarik garis.

Perintah-perintah sebagai berikut dapat mengawali sebarang perintah-perintah yang ada di depan :

- B : berpindah tapi tidak menggambar apa-apa.
- N : berpindah dan kembali ke posisi semula
- An : Membentuk sudut dengan kode n

Bila n=0 berarti 0 derajat
- 1 berarti 90 derajat
- 2 berarti 180 derajat
- 3 berarti 270 derajat

TAd : Berputar sesuai sudut d derajat, bila d positif maka putaran sesuai arah jarum jam, bila negatif maka sebaliknya.

- Cp : Mengatur warna, p berharga 0-3
- Sk : Mengatur skala k berharga 1-255
- P a,b : Mengecat dengan warna "a" pada gambar yang sebelumnya berwarna b.
- X eks : Mengerjakan ekspresi lain yang bernama eks

Statemen CIRCLE
Fungsi : untuk menggambar lingkaran.
Bentuk umum :
CIRCLE (xp,yp),r p,warnap,sa,sz[,aspek]]]
Penjelasan :
Xp,yp : koordinat pusat lingkaran
D : jari-jari lingkaran
Warna : warna gambar
Sa,sz : sudut awal dan akhir penggambaran, dapat berharga antara –2*PI dan 2*PI (PI=3.141693)
aspek : ekspresi numerik

Statemen PAINT
Fungsi : mengecat suatu daerah pada layar dengan suatu warna
Bentuk umum :
PAINT (x,y) [[,warna][.batas][,lb]]
Penjelasan :
X,y : sebarang koordinat yang harus berada dalam daerah yang akan dicat
Warna : warna pengecatan
Batas : warna batas daerah yang dicat
Lb : dipakai bila menggambar secara tile, bentuknya CHR$(&Hnn).

15. 3 Pengaturan Koordinat

Berikut ini adalah statemen-statemen yang digunakan untuk mengatur koordinat yang ada pada layar.

Statemen WINDOW
Fungsi : mendefinisikan ulang koordinat di layar dalam mode grafik
Bentuk umum :
WINDOW [[SCREEN] (x1,y1)-(x2,y2)]
Penjelasan :
SCREEN : bila ini dipakai maka titik (0,0) ada di pojok kiri atas, dan bila tidak dipakai maka titik titik (0,0) ada di pojok kanan bawah.
x1,y1 : koordinat baru untuk titik pojok kiri atas
x2,y2 : koordinat baru untuk titik pojok kanan bawah.

Statemen VIEW
Fungsi : mendefinisikan koordinat viewport pada layar
Bentuk umum :

VIEW [[SCREEN] [(x1,y1)-(x2,y2) [,[warna][.[batas]]]]]
Penjelasan :
SCREEN : bila dipakai maka koordinat statement grafik (misalnya PSET bersifat absolut), bila ini tak dipakai maka koordinat statemen bersifat relatif.

(x1,y1)-(x2,y2) : koordinat kiri atas dan kanan bawah dari pandangan yang dilihat.

warna : warna dari daerah pandangan

batas : warna dari batas daerah pandangan

Statemen POINT

Fungsi : mendeteksi warna dari suatu titik di layar

Bentuk umum :

POINT (x,y)

Penjelasan :

x , y : Koordinat titik yang akan dideteksi warnanya.

15. 4 Contoh-contoh Program

Program 1 : Menggambar segitiga

```
screen 1
line(70,80)-(130,80)
line(130,80)-(100,20)
line(100,20)-(70,80)
```

Program 2 : Menggambar kotak

```
CLS
SCREEN 1
FOR I=1 TO 10
LINE (I*20,I*20)-(I*20+50,I*20),1
LINE (I*20,I*20)-(I*20,I*20+10),2
LINE (I*20+50,I*20)-(I*20+50,I*20+10),3
LINE (I*20,I*20+10)-(I*20+50,I*20+10),1
DELAY 0.2
NEXT I
```

Program 3 : Menggambar bentuk panah

```
SCREEN 1
LINE(160,150)-(160,50)
LINE(160,50)-(130,80)
LINE(160,50)-(190,80)
```

Program 4 : Menggambar banyak panah secara acak

```
CLS
SCREEN 1
RANDOMIZE(400+I)
FOR I=1 TO 20
COLOR 3 Diktat Kuliah Bahasa Komputer I halaman 10
POSISIX=INT(RND*100+i*10)
'RANDOMIZE(150+I)
POSISIY=INT(RND*100+I*10)
LINE(POSISIX,POSISIY)-(POSISIX-15,POSISIY+10),2
LINE(POSISIX+15,POSISIY+10)-
(POSISIX,POSISIY),2
LINE(POSISIX,POSISIY)-(POSISIX,POSISIY+40),2
NEXT
```

Program 5 : Menggambar titik bergerak yang menghapus titik yang sudah ada.

```
CLS
SCREEN 1
FOR J=1 TO 15
PSET (50,10*J),J
PSET (100,10*J),J
PSET (150,10*J),J
NEXT J
DELAY 0.05
FOR I=1 TO 4
FOR J=1 TO 80
```

```
PSET(J*2,40*I-20)
DELAY 0.0275
PRESET(J*2,40*I-20)
NEXT J
NEXT I
END
```

Program 6 : Menggambar lingkaran dengan koordinat acak di layar.

```
SCREEN 1
CLS
FOR L=0 TO 400
X=INT(RND*320)
Y=INT(RND*200)
WARNA = L MOD 4
CIRCLE (X,Y),5,WARNA
NEXT
```

Program 7 : Mendeteksi warna titik di layar dari titik acak.

```
CLS
SCREEN 1
FOR J=1 TO 400
X=INT(RND*320) : Y=INT(RND*200)
WARNA=J MOD 4+1
PSET (X,Y),WARNA
DELAY 0.02
NEXT J
FOR K=1 TO 5000
X=INT(RND*320) : Y=INT(RND*200)
WARNA=POINT(X,Y) 'Mendeteksi warna di suatu titik
IF (WARNA > 0) THEN 'Ada titik berwarna
'Digambar kotak pada titik yang berwarna
```

```
LINE (X,Y) - (X+5,Y),WARNA : LINE (X,Y) -
(X,Y+5),WARNA
 LINE (X+5,Y) - (X+5,Y+5),WARNA
 LINE (X,Y+5) - (X+5,Y+5),WARNA
 END IF
 NEXT K
```

BAB XVI
GAMBAR RASTER (*BITMAP IMAGE*)

16.1 Pendahuluan

Desain grafis sering berhubungan dengan penggunaan gambar digital (digital image), yaitu gambar yang telah di-digitasi menjadi data-data digital sehingga dapat diolah dan dimanipulasi oleh komputer, tentu saja melalui perangkat lunak grafik (*graphic software*). Penggunaan yang tidak tepat pada suatu desain, selain merusak nilai estetis, juga sering menimbulkan masalah pada proses persiapan *Final Artwork* (F/A), pracetak dan pada proses cetak. Sebaliknya, penggunaan gambar yang tepat dapat meningkatkan nilai estetis desain sehingga hasil-hasil desain grafis dapat memberikan kesan mewah lux dan mewah. Pengambilan keputusan dalam pemilihan dan penggunaan gambar digital membutuhkan pengetahuan yang cukup tentang digital image, disamping faktor ketelitian seorang designer.

Dalam dunia desain grafis, dikenal 2 (dua) jenis digital image, yaitu :

- Gambar *raster* (*bitmap image*), yaitu gambar digital yang terbentuk dari sekumpulan titik penyusun gambar atau pixel (picture-x element).
- Grafik vektor (*vector graphic*), yaitu gambar digital yang terbentuk dari garis, kurva dan bidang, yang masing-masing merupakan suatu formulasi matematik.

Secara khusus tulisan ini membahas pemilihan dan penggunaan gambar raster (*bitmap image*), ada pun pemilihan dan penggunaan *grafik vektor* (*vector graphic*) akan dibahas pada tulisan yang lain.

Gambar Raster

Gambar raster (*bitmap image*) merupakan gambar digital yang tersusun dari sekumpulan titik penyusun gambar yang disebut *pixel* (*picture-x element*). *Pixel-pixel* penyusun gambar berkumpul dan bergabung membentuk seperti mozaik kemudian memanipulasi mata sehingga pada jarak pandang tertentu akan tampak kesan gambar utuh.

Gambar *raster* bersifat dependent *pixel,* artinya sangat dipengaruhi oleh banyaknya *pixel* penyusun gambar. Semakin banyak *pixel-pixel* yang menyusun gambar raster, maka kualitasnya akan semakin baik sehingga gambar terlihat halus. Sebaliknya, semakin sedikit *pixel-pixel* yang menyusun suatu gambar, maka kualitasnya akan semakin kurang sehingga gambar terlihat kasar.

Dalam pengolahan gambar *raster*, terdapat 2 (dua) hal pokok yang harus diperhatikan, yaitu ukuran gambar (*image size*) dan resolusi (*resolution*).

Ukuran Gambar (Image Size):

Ukuran gambar (*image size*) menyatakan ukuran banyaknya *pixel* penyusun gambar *raster* yang dinyatakan dalam matrik 2 dimensi, yaitu (X × Y) *Pixel*, di mana X menyatakan ukuran banyaknya *pixel* perbaris pada arah horizontal, sedangkan Y menyatakan ukuran banyaknya *pixel* perkolom pada arah vertikal. Sebagai contoh, gambar raster

berukuran 800×600 *pixel,* terdiri atas 800×600 pixel = 480.000 *pixel*, dengan susunan 800 *pixel* setiap baris pada arah horisontal dan 600 *pixel* setiap kolom pada arah vertikal.

Resolusi (Resolution)

Atribut gambar *raster* yang tak kalah pentingnya adalah resolusi (*resolution*), yang didefinisikan sebagai banyaknya *pixel* dalam setiap satuan panjang. Umumnya, resolusi dinyatakan dalam satuan dpi (dot per inchi). Sebagai contoh, gambar *raster* yang memiliki resolusi 72 dpi, berarti terdiri atas 72 dot (titik) pada setiap inchi.

Semakin tinggi resolusi suatu gambar *raster*, maka kualitasnya akan semakin baik, tetapi perlu diperhitungkan juga kemampuan mesin cetak, baik mesin cetak konvensional maupun digital printing. Gambar *raster* yang resolusinya terlalu tinggi akan menghasilkan hasil cetak yang kabur sehingga gambar *raster* tidak terlihat dengan jelas. Sebaliknya, resolusi yang terlalu rendah juga akan menghasilkan hasil cetak yang kasar. Memang terlalu rumit untuk memprediksikan dengan tepat tatawarna dan resolusi gambar *raster* pada proses cetak (press), namun dengan pengalaman dan peningkatan intensitas pekerjaan desain, kita akan menemukan tatawarna dan resolusi yang tepat pada produksi cetak (press).

Gambar 2.01 Gambar raster 640×480 pixel 72 dpi

Di atas telah dikemukakan bahwa gambar *raster* bersifat bersifat *dependent pixel* atau sangat dipengaruhi oleh jumlah *pixel* penyusun gambar. Karena sifat inilah, maka transformasi yang dilakukan pada gambar *raster*, baik memutar gambar (*rotate*) maupun memperbesar gambar (*zoom*), akan sangat berpengaruh terhadap kualitas gambar. Bahkan pada skala pembesaran (*zoom scale*) tertentu, *pixel-pixel* penyusun gambar akan terlihat jelas sehingga gambar terlihat kasar (terkesan seperti tumpukan kotak-kotak berwarna). Simaklah gambar 2.02, gambar *bitmap* berukuran 106×160 pixel, 72 dpi.

Gambar 2.02 Gambar raster 106×160 pixel, 72 dpi

Jika pada gambar 2.02 tersebut dilakukan pembesaran (*zoom*), maka kualitasnya akan menurun. Untuk membandingkannya, pada gambar 2.03 diperlihatkan hasil pembesaran (*zoom*) dengan skala pembesaran (*zoom scale*) 2 kali (200%) dari ukuran normal (Catatan, pembesaran dilakukan oleh *Photoshop*, perangkat lunak pengolah grafik).

Gambar 2.03 Gambar raster 106×160 pixel 72 dpi
Pada pembesaran 200% dengan *Photoshop*

Kelebihan / Kekurangan Gambar Raster

Setiap *pixel* penyusun gambar *raster* mampu menyimpan informasi warna dengan tajam sehingga gambar *raster* mampu menampilkan kualitas yang sangat baik. Karena kemampuan dan ketajamannya dalam menyimpan informasi warna pada setiap *pixel*, maka gambar *raster* sering digunakan untuk menyimpan photo atau gambar digital lain yang perlu ditampilkan sesuai aslinya.

Kekurangan gambar *raster* adalah sangat bergantung pada ukuran *pixel* penyusun gambar, sehingga desainer tidak bebas memperbesar/memperkecil gambar, memutar gambar, atau mentransformasikannya dalam bentuk lain

yang harus disesuaikan dengan ukuran desain yang akan dibuat.

Perangkat Lunak Pendukung Gambar Raster

Seperti dijelaskan di atas, gambar *raster* sangat dipengaruhi oleh ukuran *pixel* penyusun gambar (*dependent pixel*). Untuk itu, pengolahan *raster* harus hati-hati dan teliti. Ukuran *pixel* dan resolusi harus diperhatikan dan diperhitungkan untuk menghasilkan kualitas gambar seperti yang diharapkan. Tentu saja, perangkat lunak yang digunakan pun harus benar-benar mendukung gambar *raster*. Pemilihan perangkat lunak yang tidak tepat malah dapat merusak gambar *raster*.

Saat ini memang cukup banyak ditemukan perangkat-perangkat lunak yang mampu menampilkan gambar *raster* (*bitmap image viewer software*), seperti *ACD Systems, Corel PhotoPaint, Windows Picture*, dan sebagainya. Beberapa aplikasi yang terintegrasi pada sistem operasi, seperti *Microsoft Paint* pada Windows, juga dapat digunakan sebagai *bitmap image viewer*. Namun, tidak semua perangkat lunak mampu mengolah *bitmap* dengan baik. Masing-masing perangkat lunak memiliki kelebihan dan kelemahan dalam setiap operasi pengolahan. Jika kita hanya menginginkan kecepatan dan kemudahan untuk sekadar melihat-lihat gambar (*browsing image*), *ACD System* lebih cocok digunakan, untuk membuat gambar raster transparansi, *Corel PhotoPaint* lebih unggul, demikian seterusnya. Yang paling penting, seorang desainer harus mampu memilih dan menggunakan perangkat lunak grafik yang paling sesuai dengan kebutuhan. Pemilihan dan penggunaan perangkat lunak yang tepat akan menghasilkan desain grafis yang berkualitas.

Dalam tulisan ini, penulis banyak menggunakan aplikasi *Photoshop*, perangkat lunak pengolah grafik yang

dikembangkan oleh *Adobe Systems*. Selain memiliki fasilitas yang cukup lengkap untuk pengolahan gambar raster, *photoshop* juga mampu mengkalibrasi *color input* sehingga konfigurasi warna mulai dari perencanaan, persiapan *final artwork*, proses pra-cetak (pre-press) dan proses cetak, baik menggunakan mesin cetak konvensional maupun digital printing, dapat diprediksi dengan baik. Sebagai ilustrasi, simaklah gambar 2.04, gambar raster origital, atau gambar raster yang belum diolah dengan ukuran 640×480 pixel 72 dpi.

Gambar 2.04 Gambar raster belum diolah 640×480 pixel 72 dpi

Dengan perangkat lunak grafik yang berbeda, penulis mencoba memanipulasi atribut HSL (Hue, Saturation and Lightness) dengan konfigurasi yang sama, H(Hue) = -15%, S(Saturation) = -15% dan L(Lightness) = +15%. Ternyata hasilnya jauh berbeda, gambar 2.05 menunjukkan hasil manipulasi pada *Windows Picture*, sedangkan gambar 2.06 menunjukkan hasil manipulasi pada *Photoshop*.

Gambar 2.05 Hasil manipulasi pada *Windows Picture*

Gambar 2.06 Hasil manipulasi pada *Photoshop*

Sumber-sumber Gambar Raster

Saat ini, gambar raster dapat peroleh dari berbagai sumber, baik dari perangkat elektronik maupun yang gambar raster sudah jadi (sudah berbentuk data digital). Berikut adalah beberapa sumber gambar bitmap untuk keperluan desain grafis lengkap dengan kelebihan dan kekurangannya.

- **Pemindai (*Scanner*)**

Scanner menggunakan teknologi CCD (*Charged Couple Device*) sebagai sensor penangkap gambar. *Scanner* mampu melakukan pembesaran gambar secara optis (*optical zoom*) dengan skala pembesaran (*zoom scale*) yang sangat tergantung dari kemampuan sensor-nya. Semakin tinggi kualitas sensor pada *scanner*, maka pembesaran yang dapat dilakukan-pun semakin besar tanpa menurunkan kualitas gambar asli (gambar yang discan). Bahkan *scanner hi-end* seharga ratusan juta rupiah memiliki drum-scanner dan menerapkan teknologi PMT (*photomultiplier*) yang

dapat melakukan pembesaran gambar di atas 1000% tanpa menurunkan kualitas gambar yang asli (gambar yang di-scann).

Kelemahan *scanner* adalah kita harus memiliki data analog (photo atau gambar tercetak) sebagai sumber (master).

- **Kamera Digital**

Kamera digital sebenarnya dikembangkan dari *scanner* dengan mengeliminasi kelemahan yang dimiliki oleh *scanner*, yaitu harus adanya data analog (photo atau gambar tercetak) sebagai sumber (master). Untuk mengambil/menangkap suatu objek di lingkungan fisik (*physical environment*), kamera digital dapat langsung menyimpan informasi objek menjadi data digital sehingga tidak lagi diperlukan data analog atau gambar tercetak.

Teknologi sensor optik untuk menangkap citra warna dan cahaya pada kamera digital sama dengan teknologi sensor optik pada *scanner*, yaitu teknologi CCD (*Charged Couple Device*). Bahkan, pada pengembangan berikutnya digunakan teknologi sensor optik CMOS (*Complementary Metal Oxide Semiconductor*) untuk kamera digital dengan harga yang lebih terjangkau.

Kualitas gambar yang dihasilkan oleh kamera digital sangat dipengaruhi oleh sensor optiknya. Semakin banyak jumlah *pixel* yang dapat ditangkap oleh sensor kamera digital, maka gambar yang dihasilkan-pun akan semakin baik dengan detail yang semakin tinggi. Saat ini, teknologi CCD (*Charged Couple Device*) mampu menangkap objek sampai jutaan *pixel* (yang biasanya diukur dalam satuan *megapixel*, di mana 1 *megapixel* = 1.000.000 pixel). Bahkan, kamera digital hi-end mampu menghasilkan gambar dan merasternya dengan kapasitas lebih dari 20 MB (*megabyte)*

untuk setiap gambar. Kapasitas gambar raster sebesar ini disimpan dalam data digital berbentuk TIFF (*Tag Image File Format*), sementara kamera digital *midle-end* dan *low-end* hanya mampu menghasilkan gambar berkapasitas di bawah 5 MB untuk setiap gambar. Data digital yang disimpan-pun dikompresi (diperkecil) dalam bentuk JPEG (Joint Picture Expert Group).

Bagian utama yang membedakan kamera digital *hi-end* seharga puluhan sampai ratusan juta rupiah dibandingkan dengan kamera digital *low-end* seharga beberapa juta rupiah saja, adalah dalam kemampuan sensor optik untuk mendapatkan detail dan ketajaman gambar raster. Sebagai imbas dari pesatnya kemajuan *multiteknologi*, saat ini, teknologi kamera digital diintegrasikan pada telepon seluler. Sensor optik yang ditanamkan-pun terus ditingkatkan hingga tak menutup kemungkinan suatu saat kamera digital pada ponsel mampu menghasilkan kualitas gambar yang sangat baik.

- **Photo CD**

Untuk kemudahan mendapatkan gambar digital, saat ini banyak ditemukan jasa/layanan yang menjual gambar digital yang sudah jadi dan sudah disimpan dalam media simpan *compact disk* (CD). Dalam satu keping CD terdapat banyak gambar digital yang bisa dipilih sendiri melalui katalog atau dipesan melalui internet.

Untuk keperluan desain grafis, khususnya yang akan berhubungan dengan produksi cetak yang mementingkan kualitas, kita harus memperhatikan ukuran dan resolusi gambar pada Photo CD. Umumnya, photo CD yang tersedia saat ini dapat dikategorikan dalam 6 jenis, yaitu :

- Base 1/16
- Base 1/4

- Base
- 4 Base
- 16 Base
- 64 Base

Angka-angka yang tertera menyatakan mode ketajaman warna yang disimpan dalam Photo CD tersebut (yang dinyatakan dalam bit). Semakin tinggi tingkat ketajamannya, tentu saja kualitas gambar bitmap semakin baik sebab setiap *pixel* menyimpan informasi warna dengan detail. Untuk desainer professional dan penerapan aplikasi optimal pada perangkat lunak grafik pengolah grafik, biasanya digunakan Photo CD dengan 64 base.

- **Internet**

Internet merupakan gudangnya gambar yang dapat di-*download* atau diambil tanpa berbayar atau gratis (hanya membayar biaya pulsa internet), tetapi harus diingat bahwa gambar-gambar yang didapat dari internet umumnya memiliki kualitas yang rendah, ukuran gambar (*image size)* yang kecil dan resolusi yang rendah. Hal ini dikarenakan oleh pertimbangan kecepatan akses (*speed access*) pada internet, semakin besar gambar yang ditampilkan di internet maka kecepatan aksesnya akan semakin lambat. Para penyedia layanan internet, server dan desainer web tentu saja lebih mementingkan kecepatan akses dibandingkan kualitas gambar sehingga jarang ditemukan gambar berkualitas tinggi pada internet. Untuk keperluan desain grafis, gambar digital yang bersumber dari internet sangat tidak dianjurkan.

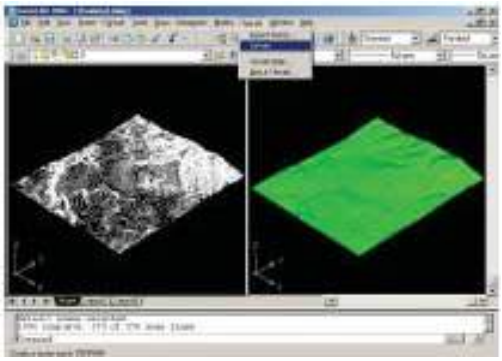
Dalam ilmu informatika dikenal suatu istilah, GIGO (*Gold In Gold Out*, *Garbage In Garbage Out*) yang artinya jika kita memasukkan emas, maka emas pulalah yang akan dihasilkan, sebaliknya jika kita memasukkan sampah, maka sampah pulalah yang akan dihasilkan. Istilah tersebut tampaknya berlaku juga dalam desain grafis, khususnya dalam pemilihan dan penggunaan gambar digital. Bagaimana mungkin akan dihasilkan hasil desain berkualitas tinggi sementara sumber gambar digitalnya sampah (berkualitas buruk). Baik atau buruknya hasil desain sangat tergantung dari keputusan designer. Seorang designer dituntut untuk teliti, cerdas, dan memiliki pengetahuan serta pengalaman yang luas, termasuk dalam penggunaan gambar raster.

BAB XVII
APLIKASI-APLIKASI KOMPUTER GRAFIS

Komputer Grafis adalah cabang dari Ilmu Komputer dan berkaitan dengan manipulasi *visual content* dan proses sistesisnya secara digital. Walaupun instilah ini sering mengacu kepada komputer grafik 3 dimensi, tetapi sebenarnya juga mencakup grafik 2 dimensi dan pengolahan citra. Saat ini, kita dapat melihat penggunaan komputer grafis di berbagai bidang dan disiplin ilmu seperti sains, keteknikan, seni, bisnis, industri, kesehatan, pemerintahan, hiburan, periklanan, pendidikan, dan masih banyak lagi. Berikut ini adalah aplikasi-aplikasi yang menggunakan komputer grafis.

17.1 *Computer-Aided Design* (CAD)

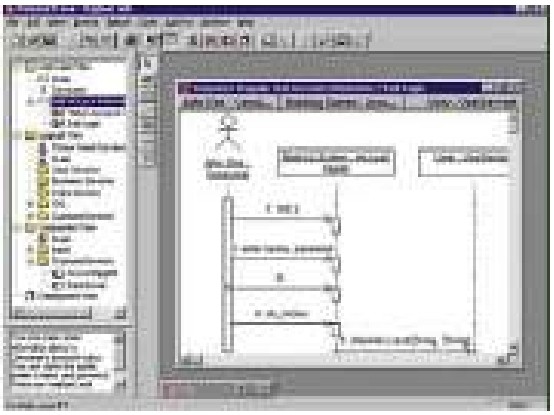
CAD adalah alat bantu berbasis komputer yang digunakan dalam proses analisis dam desain, khusunya untuk sistem arsitektural dan *engineering*. CAD banyak digunakan dalam mendesain bagunan, mobil, pesawat, komputer, alat-alat elektronik, peralatan rumah tangga, dan berbagai produk lainnya. Contoh aplikasinya: AutoCAD.



Gambar 1 Aplikasi AutoCAD salah satu aplikasi CAD.

17.2 *Computer-Aided Sofware Engineering* (CASE)

CASE mirip dengan CAD, tetapi digunakan dalam bidang *sofware engineering*. CASE digunakan dalam memodelkan *user requirement*, pemodelan basis data, *workflow* dalam proses bisnis, struktur program, dan sebagainya. Contoh aplikasi: *Rational Rose*, *SyBase Power Designer*.



Gambar 2 Rational Rose, salah satu contoh CASE.

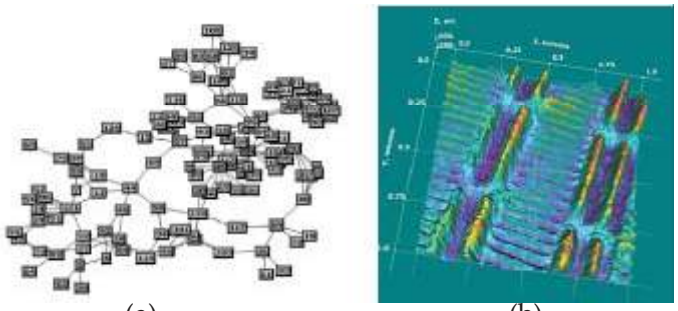
17. 3 *Virtual Reality*

Virtual Reality adalah lingkungan virtual yang seakan-akan begitu nyata di mana user dapat berinteraksi dengan objek-objek dalam suasana atau lingkungan 3 dimensi. Perangkat keras khusus digunakan untuk memberikan efek pemadangan 3 dimensi dan memampukan *user* beriteraksi dengan objek-objek yang ada dalam lingkungan. Contoh: aplikasi VR *parachute trainer* yang digunakan oleh U.S. Navy untuk latihan terjun payung. Aplikasi ini dapat memberikan keutungan berupa mengurangi resiko cedera selama latihan, mengurangi biaya penerbangan, melatih perwira sebelum melakukan terjun payung sesungguhnya.

Gambar 1 Seorang perwira U.S. Navy menggunakan VR parachute trainer

17. 4 Visualisasi Data

Visualisasi Data adalah teknik-teknik membuat image, diagram, atau animasi untuk mengkomunikasikan pesan. Visualisasi telah menjadi cara yang efektif dalam mengkomunikasikan baik data atau ide abstrak maupun nyata sejak permulaan manusia. Contoh: visualisasi dari struktur protein, strutur suatu website, visualisasi hasil data mining.

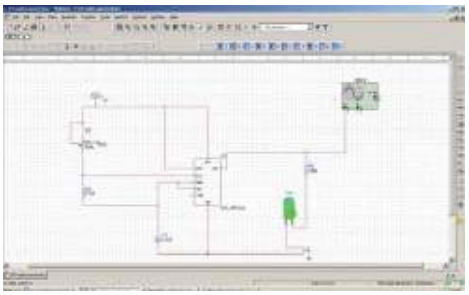


(a) (b)

Gambar 2 (a) Struktur suatu website dengan 198 hyperlink; (b) Visualisasi 3D dari gelombang magnetic pada permukaan hard drive PC

17. 5 Pendidikan dan Pelatihan

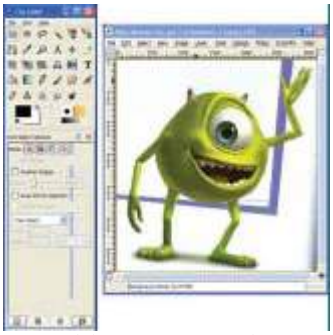
Model-model yang dihasilkan melalui komputer yang tentunya menggunakan grafis biasa digunakan sebagai alat bantu pendidikan. Model-model seperti proses-proses fisika dan kimia, fungsi-fungsi psikologi, simulasi, dan sebagainya dapat membantu sesorang memahami bagaimana operasi atau proses yang terjadi dalam suatu sistem. Contoh: simulasi rangkaian elektronik untuk pembelajaran, salah satu aplikasinya Electroni workbench.



Gambar 3 Electronic Workbench—simulasi rangkaian elektronik.

17. 6 *Computer Art*

Computer art adalah penggunaan komputer grafis untuk menghasilkan karya-karya seni. Hasil dapat berupa kartun, potret, foto, layout media cetak, logo, lukisan abstrak, desain interior atau eksterior, dan lain sebagainya. Contoh: Adobe Photoshop, Corel Painter, GIMP.



Gambar 4 GIMP—Aplikasi Image editing untuk digital art.

17. 7 Hiburan

Komputer grafis juga digunakan secara luas pada bidang *entertainment* khususnya pertelevisian, motion pictures, animasi, video clips, dan sebagainya. Film-film animasi yang berdar di pasaran seperti Shrek, Monster Inc., anime-anime Jepang, menggunakan komputer grafis.

Gambar 5 Aplikasi grafika komputer pada bidang hiburan berupa film-film animasi 3D.

17. 8 Video Game

Video game adalah permainan yang melibatkan interaksi dengan *user interface* untuk menghasilkan umpan balik berupa visualisasi pada perangkat video. Aplikasi banyak beredar di pasaran mulai yang sederhana 2 dimensi, seperti tetris, hingga yang rumit, 3 dimensi, dan memerlukan resource banyak, seperti game sepakbola Winning Eleven. Dari yang yang standalone hingga online network, seperti Ragnarok. Dari PC, console, hingga mobile devices.

Gambar 6 Video game menggunakan grafika komputer. Gambar kiri adalah permainan tertris dan gambar kanan adalah permainan sepakbola.

17. 9 Pengolahan Citra

Pengolahan citra berkaitan dengan teknik-teknik untuk modifikasi dna intepretasi citra, meningkatkan kualitas citra, analisis citra, dan mengenali pola-pola visual yang ada dalam suatu citra. Contoh: perbaikan citra sehingga menjadi lebih jelas.

(a) (b)

Gambar 7 Perbaikan kualitas citra sehingga menjadi lebih baik pada pengolahan citra digital: (a) sebelum; (b) sesudah.

17. 10 *Computer Vision*

Compute Vision adalah ilmu pengetahuan dan teknologi dari mesin-mesin yang dapat melihat. Sebagai disiplin ilmu, *computer vision* berkaitan dengan teori untuk membangun sistem buatan yang dapat menarik informasi dari citra-citra. Informasi tersebut kemudian dapat digunakan sebagai *input* dalam mengambil keputusan atau tindakan. Data citra yang diambil dapat berupa video, citra dari berbagai kamera, dan sebagainya.

17. 11 *Graphical User Interface*

Graphical User Inteface adalah antarmuka grafis yang mempermudah interaksi Manusia dengan komputer dan alat-alat yang dikendalikan oleh komputer. GUI sudah sering kita lihat berupa window-window yang digunakan pada sistem operasi Windows, Mac, maupun Linux.

Gambar 8 GUI aplikasi-aplikasi pada sistem operasi Windows XP.

BAB XVIII
MANIPULASI GRAFIK

PHP adalah salah satu bahasa pemrograman yang cukup populer dalam membuat halaman web dinamis. PHP bersifat *open source,* artinya pengguna dapat menggunakannya secara gratis dan mendistribusikannya secara bebas. JPGraph adalah salah satu library dari PHP yang berfungsi untuk membuat dan memanipulasi grafik. Dengan menggunakan JPGraph, seorang programmer dapat dengan mudah menggambar suatu grafik dengan menggunakan class-class dan method yang ada pada JPGraph. Teknik pembuatan grafik yang kedengarannya sulit dapat disederhanakan menjadi beberapa baris saja.

18. 1 Latar Belakang

JPGraph adalah *library* dari PHP yang bersifat *object oriented.* Fungsi utama dari *library* ini adalah untuk menggambar grafik pada *browser* sesuai dengan data yang ada. JPGraph bersifat *open source,* dapat didownload secara gratis melalui http://aditus.nu/JPGraph/. Perlu diperhatikan bahwa untuk menggunakan JPGraph pastikan GD Library aktif (*enable*) pada php Anda.

Terdapat perbedaan antara JPGraph yang digunakan pada php versi 5 dengan php 4, karena itu pastikan jika Anda mendowload library ini sesuai versi php yang

anda gunakan. JPGraph dapat didownload sekitar 4 MB lengkap dengan manualnya yang dapat membantu kita mempelajai isi dari JPGraph. Terdapat sekitar 87 class dan 814 method di dalam *library* ini. Di sini kita hanya akan membahas beberapa class dan metode dari JPGraph. Setelah JPGraph berhasil didownload, ekstrak file tersebut ke tempat directory kerja Anda. *Folder docs* berisi manual dari JPGraph, sedangkan folder src berisi file-file yang digunakan.

18. 2 Grafik Garis

Di sini kita akan membuat suatu grafik yang datanya disimpan dalam database MySQL. Pada database Anda buatlah tabel 'data_grafik' dengan 2 field: 'dataX' dan 'dataY' sebagai berikut:

CREATE TABLE 'data_grafik' ('dataX' int(10) default NULL, 'data Y' int(10) default NULL); Isi tabel tersebut dengan data berikut:

Data X	1	2	3	4	5	6
Data Y	10	5	8	12	6	9

Berikut ini adalah kode program untuk menampilkan data di atas ke dalam bentuk grafik garis:

```
<?
include ("JPGraph/JPGraph.php");
include ("JPGraph/JPGraph_line.php");
$db = mysql_connect("localhost", "root","") or die(mysql_
error());
mysql_select_db("test",$db) or die(mysql_error());
$sql = mysql_query("SELECT * FROM data_grafik") or
die(mysql_error());
while($row = mysql_fetch_array($sql))
{
```

```
        $data[ ] = $row[1];
        $leg[ ] = $row[0];
}
        $graph = new Graph(350,250,"auto");
        $graph->SetScale('textint');
        $graph->img->SetMargin(50,30,50,50);
        $graph->SetShadow();
        $graph->title->Set("Grafik Batang");
        $graph->xaxis->SetTickLabels($leg);
        $bplot = new LinePlot($data);
        $bplot->value->Show();
        $bplot->value->SetFont(FF_ARIAL,FS_BOLD);
        $bplot->value->SetAngle(45);
        $bplot->SetLegend("Banyak data");
        $graph->Add($bplot);
        $graph->Stroke();
?>
```

Penjelasan Kode Program

```
include ("JPGraph/JPGraph.php");
include ("JPGraph/JPGraph_line.php");
```

Baris di atas digunakan agar dapat mengakses class graph dengan jenis graph yang digunakan berbentuk garis dari library JPGraph

```
        $db = mysql_connect("localhost", "root","") or
        die(mysql_error());
        mysql_select_db("test",$db) or die(mysql_error());
        $sql = mysql_query("SELECT * FROM data_grafik")
        or die(mysql_error());
```

Baris di atas digunakan untuk koneksi MySQL dengan database ‘test’ dan nama tabel ‘data_grafik’ yang telah dibuat sebelumnya

```
        while($row = mysql_fetch_array($sql))
        {
```

```
$data[ ] = $row[1];
$leg[ ] = $row[0];
}
```

Setelah berhasil menjalankan koneksi, data field pertama($row[0] = dataX) disimpan ke dalam $leg[ ] sedangkan field kedua($row[1] = dataY) disimpan dalam $data[ ]

```
$graph = new Graph(350,250,"auto");
```

Baris di atas adalah untuk membuat *graph* baru dengan lebar=350 dan panjang=250.”auto” dimaksudkan agar proses selanjutnya dilakukan oleh *library* secara otomatis.

```
$graph->SetScale('textint');
```

Baris ini adalah menentukan tipe dari axis X dan Y yang digunakan. Axis X diset bertipe ‘text’ sedangkan axis Y bertipe ‘int’. Beberapa tipe lain yang dapat digunakan adalah *SetScale(’loglog’)* dan *SetScale(’linlog’)*

```
$graph->img->SetMargin(50,30,50,50);
```

Baris di atas untuk menentukan margin dari graph yang digunakan dengan urutan besar margin kiri, kanan, atas dan bawah *graph*.

```
$graph->SetShadow();
```

Kode Baris di atas untuk menampilkan bayangan pada graph yang akan ditampilkan $graph->title->Set("Grafik Garis");

Judul dari graph yang dibuat diberi nama “Grafik Garis”

```
$graph->xaxis->SetTickLabels($leg);
```

Axis X dari graph ditentukan dari dataX yang sebelumnya telah disimpan dalan $leg

```
$bplot = new LinePlot($data);
```

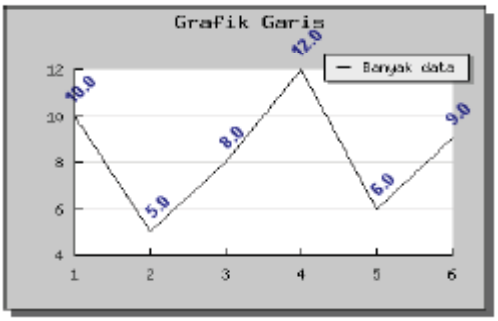
Kode baris diatas digunakan untuk membuat jenis grafik yang ditampilkan yaitu grafik garis dengan isi dataY yang telah disimpan sebelumnya di dalam $data

```
$bplot->value->Show();
```

```
$bplot->value->SetFont(FF_ARIAL,FS_BOLD);
$bplot->value->SetAngle(45);
$bplot->SetLegend("Banyak data");
```

Kode baris di atas adalah untuk menampilkan nilai dari grafik garis dengan Font yang digunakan Arial dengan tipe Bold. Nilai yang ditampilkan akan memiliki kemiringan 450 . Diberikan juga *legend* dari nilai grafik dengan nama 'Banyak data' $graph->Add($bplot);
Baris di atas untuk memasukkan grafik garis ke dalam *graph* yang telah dibuat sebelumnya. $graph->Stroke();
Kode Baris di atas diguanakn untuk menampilkan graph yang berisi grafik garis pada browser.

Bentuk grafik pada browser:



18. 3 Grafik Batang

Kita dapat mengubah jenis grafik yang digunakan dari bentuk garis menjadi grafik batang dengan mengubah baris include file kedua menjadi

```
include ("JPGraph/JPGraph_bar.php");
```

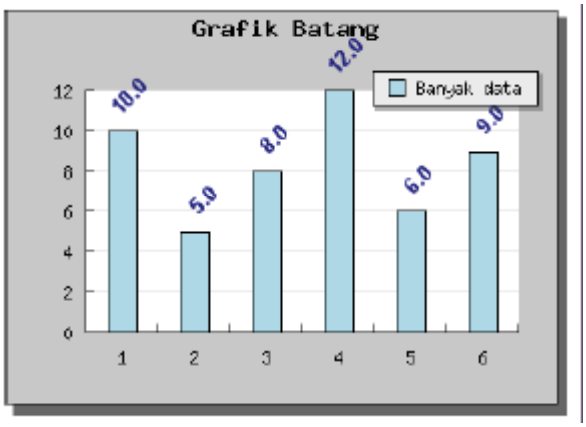
Setelah itu, dalam kode baris pembuatan grafik diganti menjadi

```
$bplot = new BarPlot($data);
```

Kode program untuk pembuatan grafik batang dengan data yang sama adalah sebagai berikut:

```
<?
include ("JPGraph/JPGraph.php");
include ("JPGraph/JPGraph_bar.php");
$db = mysql_connect("localhost", "root","") or die(mysql_
error());
mysql_select_db("test",$db) or die(mysql_error());
$sql = mysql_query("SELECT * FROM data_grafik") or
die(mysql_error());
while($row = mysql_fetch_array($sql))
{
        $data[ ] = $row[1];
        $leg[ ] = $row[0];
}
        $graph = new Graph(350,250,"auto");
        $graph->SetScale('textint');
        $graph->img->SetMargin(50,30,50,50);
        $graph->SetShadow();
        $graph->title->Set("Grafik Batang");
        $graph->xaxis->SetTickLabels($leg);
        $bplot = new BarPlot($data);
        $bplot->value->Show();
        $bplot->value->SetFont(FF_ARIAL,FS_BOLD);
        $bplot->value->SetAngle(45);
        $bplot->SetLegend("Banyak data");
        $graph->Add($bplot);
        $graph->Stroke();
?>
```

Bentuk grafik pada browser:



18. 4 Grafik PieChart 3 Dimensi

Grafik pie 3 dimensi sangat berguna untuk menampilkan data-data statistik dalam bentuk persentase dari keseluruhan data. Dibanding dengan grafik garis atau grafik batang, grafik pie memiliki tampilan yang lebih menarik dan lebih mudah dipahami pengguna. Terutama untuk data-data yang sifatnya perbandingan. Untuk data yang sama, kita dapat menampilkannya dalam bentuk grafik pie 3 dimensi. Dalam hal ini dataY dianggap sebagai banyak data yang dimiliki oleh dataX.

Berikut ini adalah kode baris yang digunakan untuk menampilkan grafik pie chart 3 dimensi

```
<?
include ("jpgraph/jpgraph.php");
include ("jpgraph/jpgraph_pie.php");
include ("jpgraph/jpgraph_pie3d.php");
$db = mysql_connect("localhost", "root","") or die(mysql_
error());
```

```
mysql_select_db(”test”,$db) or die(mysql_error());
$sql = mysql_query(”SELECT * FROM data_grafik”) or die(mysql_error());
while($row = mysql_fetch_array($sql))
{
        $data[ ] = $row[1];
        $leg[ ] = $row[0];
}
        $graph = new PieGraph(350,250,”auto”);
        $graph->SetScale(’textint’);
        $graph->img->SetMargin(50,30,50,50);
        $graph->SetShadow();
        $graph->title->Set(”Grafik Pie Chart 3 Dimensi”);
        $bplot = new PiePlot3D($data);
        $bplot->SetCenter(0.45);
        $bplot->SetLegends($leg);
        $graph->Add($bplot);
        $graph->Stroke();
?>
```

Beberapa perubahan yang terjadi antara jenis grafik sebelumnya antara lain adalah sebagai berikut:

include (”jpgraph/jpgraph_pie.php”);

include (”jpgraph/jpgraph_pie3d.php”);

Di sini ditambahkan dua *include file* untuk mengakses *class* dari *graph pie* dan grafik pie 3 dimensi.

$graph = new PieGraph(350,250,"auto");

Graph baru dibuat dengan tipe PieGraph

$bplot = new PiePlot3D($data);

Dibuat grafik dengan jenis grafik pie 3 dimensi

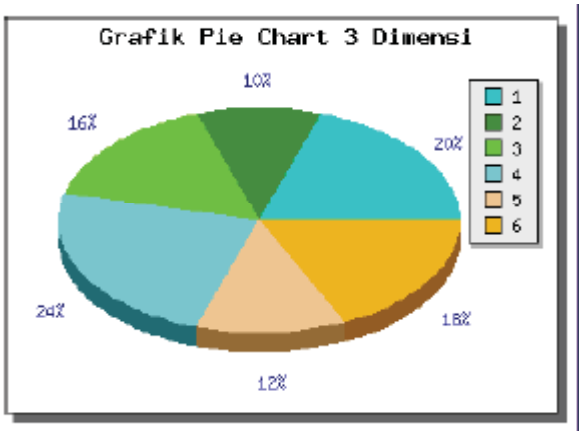
$bplot->SetCenter(0.45);

Baris di atas digunakan untuk menentukan posisi grafik pie pada graph

$bplot->SetLegends($leg);

Kode baris di atas adalah untuk menampilkan legend berdasarkan dataX($leg)

Bentuk Grafik pada Browser



BAB XIX
PENUTUP

Hal-hal yang telah dibahas di atas, hanyalah dasar-dasar dari fungsi-fungsi yang dimiliki oleh JPGraph. JPGraph masih banyak memiliki metode-metode dan *class* lainnya yang dapat membantu programmer dalam membuat grafik yang lebih baik dan menarik.

Beberapa fasilitas yang dimiliki oleh JPGraph, antara lain:

- Mendukung format grafik dengan tipe PNG, GIF maupun JPG.
- Mendukung jenis pewarnaan bergradient dengan tujuh jenis pewarnaan.
- Mendukung gambar sebagai background grafik dengan berbagai jenis.
- Mendukung jumlah grafik yang tidak terbatas dalam graph.

Hal ini memungkinkan satu *graph* memiliki berbagai macam grafik. []

DAFTAR PUSTAKA

Aho, Hopcroft, Ullman. 1987. "Data Structures and Algorithms", Prentice Hall,

Horowitz, E & Sahni, S. 1984. "Fundamentals of Data Structures in Pascal", Pitman Publishing Limited.

Knuth, D.E . 1968. "The Art of Computer Programming", Vol. 1 : "Fundamentals Algoritms", Addison Wisley.

Knuth, D.E. 1968. "The Art of Computer Programming", Vol. 3 : "Sorting and Searching", Addison Wisley, 1971

Meyer and Baudoin. 1980. "Methodes de Programmation", Eyrolles.

Scholl P.C and Peyrin, J.P. 1988. "Schemas Algorithmiques Fondamentaux", Masson.

Sedgewick R. 1975. "Algorithms", Addison Wisley, 1984.

Wirth, N. : "Systematic programming", Prentice Hall.

Wirth, N. 1986. "Algorithms & Data Stuctures", Prentice Hall.

Djikstra, E. 1989. "On the Cruelty of Really Teaching Computer Science", The

SIGCSE Award Lecturer, SIGCSE Bulletin, vol.21, no.1, Feb, pp xxiv -xxxix

Hoare, C.A.R. 1984. "Programming: Sorcery or Science ?", IEEE software, April, pp 6 - 16

Knuth, D. E. "Programming As an Art", Communication of the ACM, Vol. 17, No. 12, Dec. 74, pp 667 – 673

http://www.phpfreaks.com/tutorials/112/

Arief Ramadhan, Hendra Saputra. 2005. *PHP5 dn MySQL*, Elex Media Komputindo.

DAFTAR ISTILAH

- *Alphamosaic* adalah sebuah *text video* yang menampilkan layar terpisah menjadi garis yang tidak terlihat, dan menerima sinyal yang memerintahkan bagaimana setiap kotak terisi, seperti membuat mosaic.
- *Bump* adalah suatu teknik yang digunakan untuk menambahkan realisme dari permukaaan dengan cara mengubah refleksi dari cahaya dari permukaan tersebut.
- *Cartoon physics* adalah suatu persamaan yang mengacu kepada fakta bahwa animasi membolehkan peraturan ilmu alam yang biasa untuk diabaikan dengan cara yang humoris, untuk *effect* yang lebih dramatis.
- *Metropolis light transfer* adalah suatu prosedur yang membangun jalan dari mata kepada sumber cahaya menggunakan dua arah, kemudian membangun sedikit modifikasi ke arah yang seharusnya.
- *Euler* adalah sebuah bahasa pemrograman diciptakan oleh *niklaus witrh* dan *helmut weber*, mengandung arti sebagai pengumuman dan penyamarataan.
- *Frame rate* adalah frekuensi di mana *frame* dari *video* ditampilkan di monitor. Biasanya dijelaskan dengan *frames per second*.

- *Gouraud Shading* adalah proses di mana informasi tentang warna di interpolasikan sepanjang tampak muka dari *polygon* untuk menentukan warna di setiap *pixel*.
- *Kinematics* adalah cabang dari mekanik yang berhadapan dengan deskripsi dari pergerakan tubuh atau cairan tanpa referensi untuk memaksakan memproduksi pergerakan.
- *LATENCY* adalah persamaaan dari penundaan suatu *expressi* dari berapa banyak waktu yang dibutuhkan dari sebuah paket data untuk mengambil dari satu *point* yang ditunjuk ke lainnya.
- *Motion blur* adalah lintasan yang tampak dari benda yang bergerak cepat dalam gambar fragmen atau urutan dari gambar seperti film atau animasi.
- *New Age Healer* adalah penggunaan dari warna untuk menyeimbangkan energi di manapun tubuh kita kekurangan, seperti kekurangan atau keletihan, secara emosi, secara batiniah, atau mental. Ini berdasarkan asas dasar bahwa setiap organ dan system tubuh mempunyai karakteristik sendiri dan energi getaran, dan penyakit dapat disembuhkan dengan cara menerapkan warna dari energi getaran yang bersangkutan.
- *nVidia* adalah pembuat alat grapis. Tidak menjual dalam satu *board*, tetapi lebih menjual *chipset* -nya ke penjual pihak ke tiga untuk dijadikan satu dan dijual kembali.
- *Opaque* adalah tidak memancarkan atau memantulkan cahaya atau energi panas, yang tidak dapat dilihat.
- PMS (*Pantone Matching System*) adalah sebuah nama yang terdaftar untuk tinta berwarna system yang mempertemukan warna.

- *Photon mapping* adalah algoritma penerangan secara umum yang dikembangkan oleh Henrik Wann Jensenn. Yang bisa menyelesaikan persamaan render.
- *Radiosity* adalah cara baru untuk merender gambar tiga dimensi. Cara ini menirukan energi ringan yang berlimpah di dunia.
- *Real-time* adalah mengadakan dengan seketika. Untuk kartu redit process, ini artinya bahwa ketepatan data dari *credit card* pelanggan.
- *Refresh rate* adalah suatu jumlah waktu dalam hitungan detik yang dapat menampilkan gambar berupa data dalam waktu yang ditentukan.
- *Rendering* adalah proses membuat gambar dari model, menggunakan program komputer. Model tersebut adalah penjelasan dari objek tiga dimensi dalam bahasa yang telah ditentukan.
- *Rodmap* adalah suatu waktu perjalanan untuk menentukan bagian dari program yang spesifik untuk bisa dibuka dengan panggilan untuk mengemukakan dan batas waktu.
- *Numerical analysis* adalah bagian penting dari instruksi dan secara tegas menggunakan metode pengulangan untuk memperkirakan waktu dari solusi.
- SIGGRAPH (*Special Interest Group on GRAPHics and Interactive Techniques*) adalah nama dari pertemuan tahunan dalam *graphic computer* diadakan oleh organisasi ACM SIGGRAPH.
- *Sketchpad* adalah sebuah program komputer yang revolusioner ditulis oleh Ivan Sutherland (1938) pada tahun 1963 dalam proses penyelesaian tesisnya.
- *Text-stroke-color* adalah menetukan warna yang dipakai untuk mencoret tulisan.

- *Temporal aliasing* adalah sebuah istilah yang berhubungan dengan penampakan gejala, atau lebih dikenal dengan *stroboscopic effect*.
- *WYSIAYG* adalah deskripsi dari tampilan program komputer. Kepanjangan dari istilah tersebut adalah "what you see is all you get" .