BAB I
PENGANTAR ALGORITMA DAN PROGRAM

1.1. Apakah Itu Algoritma

Ditinjau dari asal-usul katanya, 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 namun 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*.

1.1.1. 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.

Dalam beberapa konteks, algoritma adalah spesifikasi urutan langkah untuk melakukan pekerjaan tertentu. Pertimbangan dalam pemilihan algoritma adalah, pertama, algoritma haruslah benar. Artinya algoritma akan memberikan keluaran yang dikehendaki dari sejumlah masukan yang diberikan. Tidak peduli sebagus apapun algoritma, kalau memberikan keluaran yang salah, pastilah algoritma tersebut bukanlah algoritma yang baik.

Pertimbangan kedua yang harus diperhatikan adalah kita harus mengetahui seberapa baik hasil yang dicapai oleh algoritma tersebut. Hal ini penting terutama pada algoritma untuk menyelesaikan masalah yang memerlukan aproksimasi hasil (hasil yang hanya berupa pendekatan). Algoritma yang baik harus mampu memberikan hasil yang sedekat mungkin dengan nilai yang sebenarnya.

Ketiga adalah efisiensi algoritma. Efisiensi algoritma dapat ditinjau dari 2 hal yaitu efisiensi waktu dan memori. Meskipun algoritma memberikan keluaran yang benar (paling mendekati), tetapi jika kita harus menunggu berjam-jam untuk mendapatkan keluarannya, algoritma tersebut biasanya tidak akan dipakai, setiap orang menginginkan keluaran yang cepat. Begitu juga dengan memori, semakin besar memori yang terpakai maka semakin buruklah algoritma tersebut.

Dalam kenyataannya, setiap orang bisa membuat algoritma yang berbeda untuk menyelesaikan suatu permasalahan, walaupun terjadi perbedaan dalam menyusun algoritma, tentunya kita mengharapkan keluaran yang sama. Jika terjadi demikian, carilah algoritma yang paling efisien dan cepat.

1.1.2. Beda Algoritma dan Program

Program adalah kumpulan pernyataan komputer, sedangkan metode dan tahapan sistematis dalam program adalah algoritma. Program ditulis dengan menggunakan bahasa pemrograman. Jadi bisa disebut bahwa program adalah suatu implementasi dari bahasa pemrograman.

Beberapa pakar memberi formula bahwa:

Program = Algoritma + Bahasa (Struktur Data)

Bagaimanapun juga struktur data dan algoritma berhubungan sangat erat pada sebuah program. Algoritma yang baik tanpa pemilihan struktur data yang tepat akan membuat program menjadi kurang baik, demikian juga sebaliknya.

Pembuatan algoritma mempunyai banyak keuntungan di antaranya:

1. Pembuatan atau penulisan algoritma tidak tergantung pada bahasa pemrograman manapun, artinya penulisan algoritma independen dari bahasa pemrograman dan komputer yang melaksanakannya.
2. Notasi algoritma dapat diterjemahkan ke dalam berbagai bahasa pemrograman.
3. Apapun bahasa pemrogramannya, *output* yang akan dikeluarkan sama karena algoritmanya sama.

Beberapa hal yang perlu diperhatikan dalam membuat algoritma:

1. Teks algoritma berisi deskripsi langkah-langkah penyelesaian masalah. Deskripsi tersebut dapat ditulis dalam notasi apapun asalkan mudah dimengerti dan dipahami.
2. Tidak ada notasi yang baku dalam penulisan teks algoritma seperti notasi bahasa pemrograman. Notasi yang digunakan dalam menulis algoritma disebut notasi algoritmik.
3. Setiap orang dapat membuat aturan penulisan dan notasi algoritmik sendiri. Hal ini dikarenakan teks algoritma tidak sama dengan teks program. Namun, supaya notasi algoritmik mudah ditranslasikan ke dalam notasi bahasa pemrograman tertentu, maka sebaiknya notasi algoritmik tersebut berkorespondensi dengan notasi bahasa pemrograman secara umum.
4. Notasi algoritmik bukan notasi bahasa pemrograman, karena itu *pseudocode* dalam notasi algoritmik tidak dapat dijalankan oleh komputer. Agar dapat dijalankan oleh komputer, *pseudocode* dalam notasi algoritmik harus ditranslasikan atau diterjemahkan ke dalam notasi bahasa pemrograman yang dipilih. Perlu diingat bahwa orang yang menulis program sangat terikat dalam aturan tata bahasanya dan spesifikasi mesin yang menjalannya.
5. Algoritma sebenarnya digunakan untuk membantu kita dalam mengkonversikan suatu permasalahan ke dalam bahasa pemrograman.
6. Algoritma merupakan hasil pemikiran konseptual, supaya dapat dilaksanakan oleh komputer, algoritma harus ditranslasikan ke dalam notasi bahasa pemrograman. Ada beberapa hal yang harus diperhatikan pada translasi tersebut, yaitu:
   a. Pendeklarasian variabel
      Untuk mengetahui dibutuhkannya pendeklarasian variabel dalam penggunaan bahasa pemrograman apabila tidak semua bahasa pemrograman membutuhkannya.

b. Pemilihan tipe data
   Apabila bahasa pemrograman yang akan digunakan membutuhkan pendeklarasian variabel maka perlu hal ini dipertimbangkan pada saat pemilihan tipe data.
c. Pemakaian instruksi-instruksi
   Beberapa instruksi mempunyai kegunaan yang sama tetapi masing-masing memiliki kelebihan dan kekurangan yang berbeda.
d. Aturan sintaksis
   Pada saat menuliskan program kita terikat dengan aturan sintaksis dalam bahasa pemrograman yang akan digunakan.
e. Tampilan hasil
   Pada saat membuat algoritma kita tidak memikirkan tampilan hasil yang akan disajikan. Hal-hal teknis ini diperhatikan ketika mengkonversikannya menjadi program.
f. Cara pengoperasian *compiler* atau *interpreter*.
   Bahasa pemrograman yang digunakan termasuk dalam kelompok *compiler* atau *interpreter*.

1.1.3. Algoritma Merupakan Jantung Ilmu Informatika

Algoritma adalah jantung ilmu komputer atau informatika. Banyak cabang ilmu komputer yang mengarah ke dalam terminologi algoritma. Namun, jangan beranggapan algoritma selalu identik dengan ilmu komputer saja. Dalam kehidupan sehari-hari pun 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-langkah 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.

Algoritma adalah deskripsi dari suatu pola tingkah laku yang dinyatakan secara primitif yaitu aksi-aksi yang didefenisikan sebelumnya dan diberi nama, dan diasumsikan sebelumnya bahwa aksi-aksi tersebut dapat kerjakan sehingga dapat menyebabkan kejadian.

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:
1. Mengerti setiap langkah dalam algoritma.
2. Mengerjakan operasi yang bersesuaian dengan langkah tersebut.

Tabel 1.1. Contoh-Contoh Algoritma dalam Kehidupan Sehari-hari

No.	Proses	Algoritma	Contoh Langkah dalam Algoritma
1	Membuat kue	Resep kue	Masukkan telur ke dalam wajan, kocok sampai mengembang
2	Membuat pakaian	Pola pakaian	Gunting kain dari pinggir kiri bawah ke arah kanan sejauh 5 cm
3	Merakit mobil	Panduan merakit	Sambungkan komponen A dengan komponen B
4	Kegiatan sehari-hari	Jadwal harian	Pukul 06.00: mandi pagi, pukul 07.00: berangkat kuliah
5	Mengisi voucer HP	Panduan pengisian	Tekan 888, masukkan nomor voucer

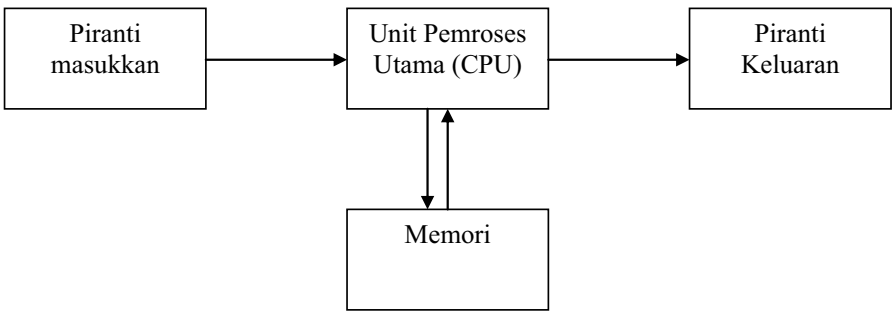
1.1.4. Mekanisme Pelaksanaan Algoritma oleh Pemroses

Komputer hanyalah salah satu pemroses. Agar dapat dilaksanakan oleh komputer, algoritma harus ditulis dalam notasi bahasa pemrograman sehingga dinamakan program. Jadi program adalah perwujudan atau implementasi teknis algoritma yang ditulis dalam bahasa pemrograman tertentu sehingga dapat dilaksanakan oleh komputer.

Kata “algoritma” dan “program” seringkali dipertukarkan dalam penggunaannya. Misalnya ada orang yang berkata seperti ini: “program pengurutan data menggunakan algoritma *selection sort*”. Atau pertanyaan seperti ini: “bagaimana algoritma dan program menggambarkan grafik tersebut?”. Jika Anda sudah memahami pengertian algoritma yang sudah

disebutkan sebelum ini, Anda dapat membedakan arti kata algoritma dan program. Algoritma adalah langkah-langkah penyelesaikan masalah, sedangkan program adalah realisasi algoritma dalam bahasa pemrograman. Program ditulis dalam salah satu bahasa pemrograman dan kegiatan membuat program disebut **pemrograman** (*programming*). Orang yang menulis program disebut **pemrogram** (*programmer).* Tiap-tiap langkah di dalam program disebut **pernyataan** atau **instruksi.** Jadi, program tersusun atas sederetan instruksi. Bila suatu instruksi dilaksanakan, maka operasi-operasi yang bersesuaian dengan instruksi tersebut dikerjakan komputer.

Secara garis besar komputer tersusun atas empat komponen utama yaitu, piranti masukan, piranti keluaran, unit pemroses utama, dan memori. Unit pemroses utama (*Central Processing Unit – CPU)* adalah "otak" komputer, yang berfungsi mengerjakan operasi-operasi dasar seperti operasi perbandingan, operasi perhitungan, operasi membaca, dan operasi menulis. Memori adalah komponen yang berfungsi menyimpan atau mengingat-ingat. Yang disimpan di dalam memori adalah program (berisi operasi-operasi yang akan dikerjakan oleh CPU) dan data atau informasi (sesuatu yang diolah oleh operasi-operasi). Piranti masukan dan keluaran (*I/O devices)* adalah alat yang memasukkan data atau program ke dalam memori, dan alat yang digunakan komputer untuk mengkomunikasikan hasil-hasil aktivitasnya. Contoh piranti masukan antara lain, papan kunci (*keyboard)*, pemindai (*scanner)*, dan cakram (*disk).* Contoh piranti keluaran adalah, layar peraga (*monitor)*, pencetak (*printer)*, dan cakram.



Gambar 1.1 Komponen-Komponen Utama Komputer

Mekanisme kerja keempat komponen di atas dapat dijelaskan sebagai berikut. Mula-mula program dimasukkan ke dalam memori komputer. Ketika program dilaksanakan (*execute)*, setiap instruksi yang telah

tersimpan di dalam memori dikirim ke CPU. CPU mengerjakan operasi-operasi yang bersesuaian dengan instruksi tersebut. Bila suatu operasi memerlukan data, data dibaca dari piranti masukan, disimpan di dalam memori lalu dikirim ke CPU untuk operasi yang memerlukannya tadi. Bila proses menghasilkan keluaran atau informasi, keluaran disimpan ke dalam memori, lalu memori menuliskan keluaran tadi ke piranti keluaran (misalnya dengan menampilkannya di layar monitor).

1.1.5. Belajar Memprogram dan Belajar Bahasa Pemrograman

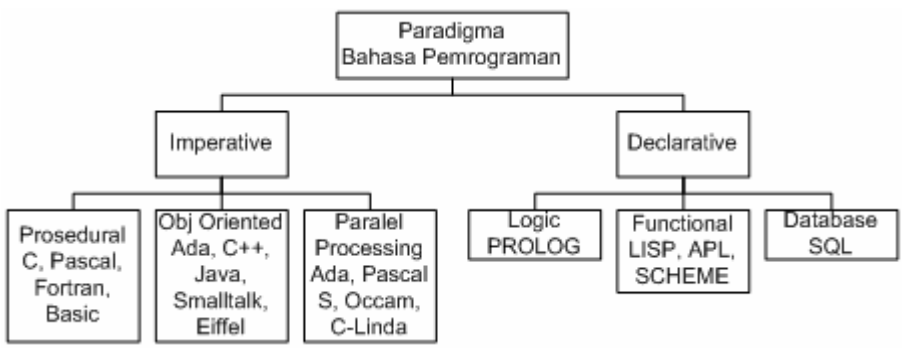
Belajar memprogram tidak sama dengan belajar bahasa pemrograman. Belajar memprogram adalah belajar tentang metodologi pemecahan masalah, kemudian menuangkannya dalam suatu notasi tertentu yang mudah dibaca dan dipahami. Sedangkan belajar bahasa pemrograman berarti belajar memakai suatu bahasa aturan-aturan tata bahasanya, pernyataan-pernyataannya, tata cara pengoperasian *compiler*-nya, dan memanfaatkan pernyataan-pernyataan tersebut untuk membuat program yang ditulis hanya dalam bahasa itu saja. Sampai saat ini terdapat puluhan bahasa pemrogram, antara lain bahasa rakitan (*assembly), Fortran, Cobol, Ada, PL/I, Algol, Pascal, C, C++, Basic, Prolog, LISP, PRG*, bahasa-bahasa simulasi *seperti CSMP, Simscript, GPSS, Dinamo*. Berdasarkan terapannya, bahasa pemrograman dapat digolongkan atas dua kelompok besar:

1. Bahasa pemrograman bertujuan khusus. Yang termasuk kelompok ini adalah *Cobol* (untuk terapan bisnis dan administrasi). *Fortran* (terapan komputasi ilmiah), bahasa rakitan (terapan pemrograman mesin), *Prolog* (terapan kecerdasan buatan), bahasa-bahasa simulasi, dan sebagainya.
2. Bahasa perograman bertujuan umum, yang dapat digunakan untuk berbagai aplikasi. Yang termasuk kelompok ini adalah bahasa *Pascal, Basic* dan *C*. Tentu saja pembagian ini tidak kaku. Bahasa-bahasa bertujuan khusus tidak berarti tidak bisa digunakan untuk aplikasi lain. *Cobol* misalnya, dapat juga digunakan untuk terapan ilmiah, hanya saja kemampuannya terbatas. Yang jelas, bahasa-bahasa pemrograman yang berbeda dikembangkan untuk bermacam-macam terapan yang berbeda pula.

Berdasarkan pada apakah notasi bahasa pemrograman lebih “dekat” ke mesin atau ke bahasa manusia, maka bahasa pemrograman dikelompokkan atas dua macam:

1. Bahasa tingkat rendah. Bahasa jenis ini dirancang agar setiap instruksinya langsung dikerjakan oleh komputer, tanpa harus melalui penerjemah (*translator*). Contohnya adalah bahasa mesin. CPU mengambil instruksi dari memori, langsung mengerti dan langsung mengerjakan operasinya. Bahasa tingkat rendah bersifat primitif, sangat sederhana, orientasinya lebih dekat ke mesin, dan sulit dipahami manusia. Sedangkan bahasa rakitan dimasukkan ke dalam kelompok ini karena alasan notasi yang dipakai dalam bahasa ini lebih dekat ke mesin, meskipun untuk melaksanakan instruksinya masih perlu penerjemahan ke dalam bahasa mesin.
2. Bahasa tingkat tinggi, yang membuat pemrograman lebih mudah dipahami, lebih "manusiawi", dan berorientasi ke bahasa manusia (bahasa Inggris). Hanya saja, program dalam bahasa tingkat tinggi tidak dapat langsung dilaksanakan oleh komputer. Ia perlu diterjemahkan terlebih dahulu oleh sebuah *translator bahasa* (yang disebut kompilator atau *compiler)* ke dalam bahasa mesin sebelum akhirnya dieksekusi oleh CPU. Contoh bahasa tingkat tinggi adalah *Pascal, PL/I, Ada, Cobol, Basic, Fortran, C, C++,* dan sebagainya.

Bahasa pemrograman bisa juga dikelompokkan berdasarkan pada tujuan dan fungsinya. Di antaranya adalah:



Gambar 1.2 Pembagian Bahasa Pemrograman

Secara sistematis berikut diberikan kiat-kiat untuk belajar memprogram dan belajar bahasa pemrograman serta produk yang dapat dihasilkan:

a. 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.

b. Belajar Bahasa Pemrograman

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

c. 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.

1.2. Menilai Sebuah Algoritma

Ketika manusia berusaha memecahkan masalah, metode atau teknik yang digunakan untuk memecahkan masalah itu ada kemungkinan bisa banyak (tidak hanya satu). Dan kita memilih mana yang terbaik di antara teknik-teknik itu. Hal ini sama juga dengan algoritma, yang memungkinkan suatu permasalahan dipecahkan dengan metode dan logika yang berlainan. Yang menjadi pertanyaan adalah bagaimana mengukur mana algoritma yang terbaik?

Beberapa persyaratan untuk menjadi algoritma yang baik adalah:

- Tingkat kepercayaannya tinggi (*realibility*). Hasil yang diperoleh dari proses harus berakurasi tinggi dan benar.
- Pemrosesan yang efisien (*cost* rendah). Proses harus diselesaikan secepat mungkin dan frekuensi kalkulasi yang sependek mungkin.

- Sifatnya general. Bukan sesuatu yang hanya untuk menyelesaikan satu kasus saja, tapi juga untuk kasus lain yang lebih general.
- Bisa dikembangkan (*expandable*). Haruslah sesuatu yang dapat kita kembangkan lebih jauh berdasarkan perubahan *requirement* yang ada.
- Mudah dimengerti. Siapapun yang melihat, dia akan bisa memahami algoritma Anda. Susah dimengertinya suatu program akan membuat susah di-*maintenance* (kelola).
- Portabilitas yang tinggi (*portability*). Bisa dengan mudah diimplementasikan di berbagai *platform* komputer.
- *Precise* (tepat, betul, teliti). Setiap instruksi harus ditulis dengan seksama dan tidak ada keragu-raguan, dengan demikian setiap instruksi harus dinyatakan secara eksplisit dan tidak ada bagian yang dihilangkan karena pemroses dianggap sudah mengerti. Setiap langkah harus jelas dan pasti.
  Contoh: Tambahkan 1 atau 2 pada x.
  Instruksi di atas terdapat keraguan.
- Jumlah langkah atau instruksi berhingga dan tertentu. Artinya, untuk kasus yang sama banyaknya, langkah harus tetap dan tertentu meskipun datanya berbeda.
- Efektif. Tidak boleh ada instruksi yang tidak mungkin dikerjakan oleh pemroses yang akan menjalankannya.
  Contoh: Hitung akar 2 dengan presisi sempurna.
  Instruksi di atas tidak efektif, agar efektif instruksi tersebut diubah. Misal: Hitung akar 2 sampai lima digit di belakang koma.
- Harus *terminate*. Jalannya algoritma harus ada kriteria berhenti. Pertanyaannya adalah apakah bila jumlah instruksinya berhingga maka pasti *terminate*?
- *Output* yang dihasilkan tepat. Jika langkah-langkah algoritmanya logis dan diikuti dengan seksama maka dihasilkan *output* yang diinginkan.

1.3. Penyajian Algoritma

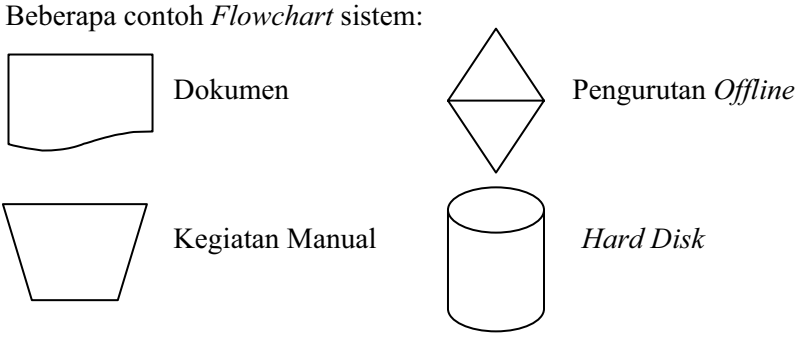
Penyajian algoritma secara garis besar bisa dalam 2 bentuk penyajian yaitu tulisan dan gambar. Algoritma yang disajikan dengan tulisan yaitu dengan struktur bahasa tertentu (misalnya bahasa Indonesia atau bahasa Inggris) dan *pseudocode*. *Pseudocode* adalah kode yang mirip dengan kode pemrograman yang sebenarnya seperti Pascal, atau C, sehingga lebih tepat digunakan untuk menggambarkan algoritma yang akan dikomunikasikan kepada pemrogram. Sedangkan algoritma disajikan dengan gambar,

misalnya dengan *flowchart*. Secara umum, *pseudocode* mengekspresikan ide-ide secara informal dalam proses penyusunan algoritma. Salah satu cara untuk menghasilkan kode pseudo adalah dengan meregangkan aturan-aturan bahasa formal yang dengannya versi akhir dari algoritma akan diekspresikan. Pendekatan ini umumnya digunakan ketika bahasa pemrograman yang akan digunakan telah diketahui sejak awal.

Flowchart merupakan gambar atau bagan yang memperlihatkan urutan dan hubungan antar proses beserta pernyataannya. Gambaran ini dinyatakan dengan simbol. Dengan demikian setiap simbol menggambarkan proses tertentu. Sedangkan antara proses digambarkan dengan garis penghubung. Dengan menggunakan *flowchart* akan memudahkan kita untuk melakukan pengecekan bagian-bagian yang terlupakan dalam analisis masalah. Di samping itu *flowchart* juga berguna sebagai fasilitas untuk berkomunikasi antara pemrogram yang bekerja dalam tim suatu proyek.

Ada dua macam *flowchart* yang menggambarkan proses dengan komputer, yaitu:

1. *Flowchart* **sistem** yaitu bagan dengan simbol-simbol tertentu yang menggambarkan urutan prosedur dan proses suatu *file* dalam suatu media menjadi *file* di dalam media lain, dalam suatu sistem pengolahan data.
   Beberapa contoh *Flowchart* sistem:



2. *Flowchart* **program** yaitu bagan dengan simbol-simbol tertentu yang menggambarkan urutan proses dan hubungan antar proses secara mendetail di dalam suatu program.

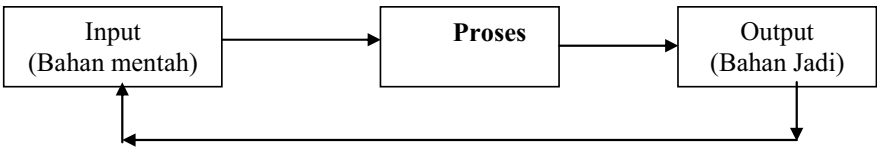
Kaidah-Kaidah Umum Pembuatan *Flowchart* Program

Dalam pembuatan *flowchart* Program tidak ada rumus atau patokan yang bersifat mutlak. Karena *flowchart* merupakan gambaran hasil pemikiran dalam menganalisis suatu masalah dengan komputer. Sehingga *flowchart*

yang dihasilkan dapat bervariasi antara satu pemrogram dengan yang lainnya.

Namun secara garis besar setiap pengolahan selalu terdiri atas 3 bagian utama, yaitu:

- *Input*,
- Proses pengolahan dan
- *Output*



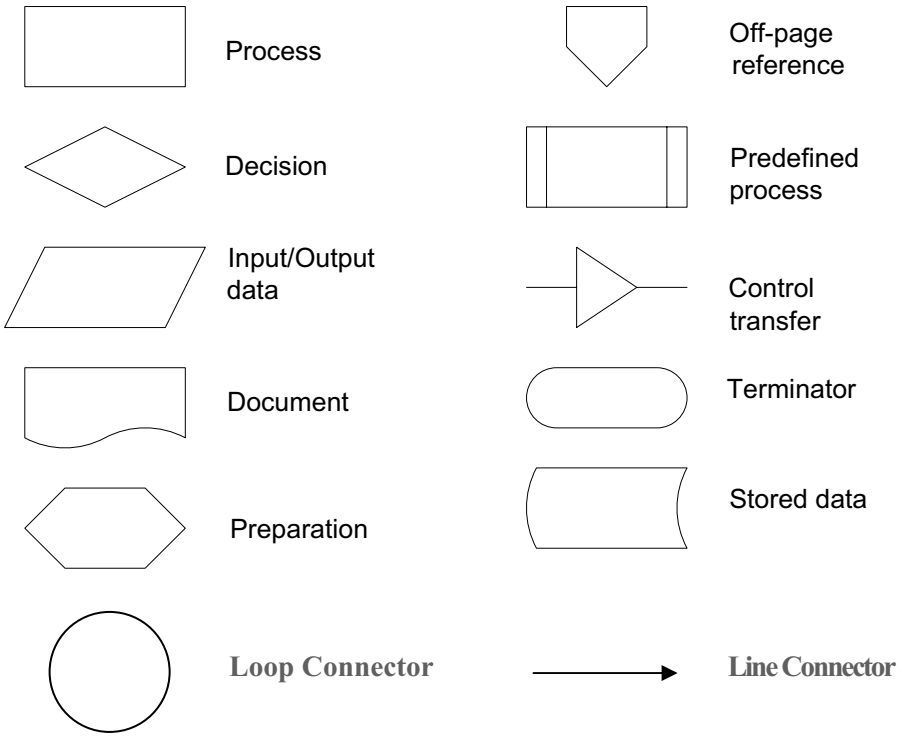
Untuk pengolahan data dengan komputer, urutan dasar pemecahan suatu masalah:

- START, berisi pernyataan untuk persiapan peralatan yang diperlukan sebelum menangani pemecahan persoalan.
- READ, berisi pernyataan kegiatan untuk membaca data dari suatu peralatan *input.*
- PROSES, berisi kegiatan yang berkaitan dengan pemecahan persoalan sesuai dengan data yang dibaca.
- WRITE, berisi pernyataan untuk merekam hasil kegiatan ke peralatan *output.*
- END, mengakhiri kegiatan pengolahan.

Walaupun tidak ada kaidah-kaidah yang baku dalam penyusunan *flowchart*, namun ada beberapa anjuran:

- Hindari pengulangan proses yang tidak perlu dan logika yang berbelit sehingga jalannya proses menjadi singkat.
- Jalannya proses digambarkan dari atas ke bawah dan diberikan tanda panah untuk memperjelas.
- Sebuah *flowchart* diawali dari satu titik START dan diakhiri dengan END.

Berikut merupakan beberapa contoh simbol *flowchart* yang disepakati oleh dunia pemrograman:



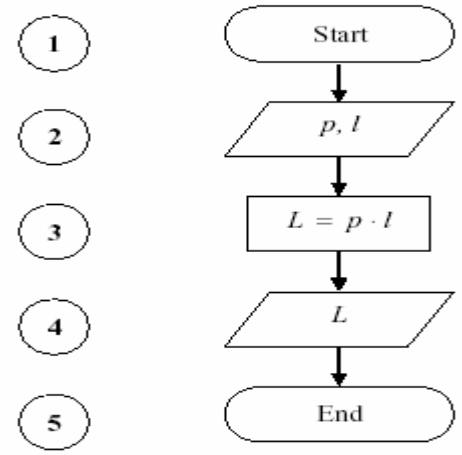
Untuk memahami lebih dalam mengenai *flowchart* ini, akan diambil sebuah kasus sederhana.

Kasus:
Buatlah sebuah rancangan program dengan menggunakan *flowchart*, mencari luas persegi panjang.

Solusi:
Perumusan untuk mencari luas persegi panjang adalah:

$$L = p.\ l$$

di mana, ***L*** adalah Luas persegi panjang, ***p*** adalah panjang persegi, dan ***l*** adalah lebar persegi.



Keterangan 1:

1. Simbol pertama menunjukkan dimulainya sebuah program.
2. Simbol kedua menunjukkan bahwa *input* data dari p dan l.
3. Data dari *p* dan *l* akan diproses pada simbol ketiga dengan menggunakan perumusan *L = p. l*
4. Simbol keempat menunjukkan hasil *output* dari proses dari simbol ketiga.
5. Simbol kelima atau terakhir menunjukkan berakhirnya program dengan tanda *End.*

1.4. Struktur Dasar Algoritma

Algoritma berisi langkah-langkah penyelesaian suatu masalah. Langkah-langkah tersebut dapat berupa runtunan aksi (*sequence*), pemilihan aksi (*selection*), pengulangan aksi (*iteration*) atau kombinasi dari ketiganya. Jadi struktur dasar pembangunan algoritma ada tiga, yaitu:

1. Struktur Runtunan
   Digunakan untuk program yang pernyataannya *sequential* atau urutan.
2. Struktur Pemilihan
   Digunakan untuk program yang menggunakan pemilihan atau penyeleksian kondisi.
3. Struktur Perulangan
   Digunakan untuk program yang pernyataannya akan dieksekusi berulang-ulang.

1.5. Tahapan dalam Pemrograman

Langkah-langkah yang dilakukan dalam menyelesaikan masalah dalam pemrograman dengan komputer adalah:

1. Definisikan Masalah
   Berikut adalah hal-hal yang harus diketahui dalam analisis masalah supaya kita mengetahui bagaimana permasalahan tersebut:
   a. Kondisi awal, yaitu *input* yang tersedia.
   b. Kondisi akhir, yaitu *output* yang diinginkan.
   c. Data lain yang tersedia.
   d. Operator yang tersedia.
   e. Syarat atau kendala yang harus dipenuhi.

   Contoh kasus:
   Menghitung biaya percakapan telepon di wartel. Proses yang perlu diperhatikan adalah:
   a. *Input* yang tersedia adalah jam mulai bicara dan jam selesai bicara.
   b. *Output* yang diinginkan adalah biaya percakapan.
   c. Data lain yang tersedia adalah besarnya pulsa yang digunakan dan biaya per pulsa.
   d. Operator yang tersedia adalah pengurangan (-), penambahan (+), dan perkalian (*).
   e. Syarat kendala yang harus dipenuhi adalah aturan jarak dan aturan waktu.

2. Buat Algoritma dan Struktur Cara Penyelesaian
   Jika masalahnya kompleks, maka dibagi ke dalam modul-modul. Tahap penyusunan algoritma seringkali dimulai dari langkah yang global terlebih dahulu. Langkah global ini diperhalus sampai menjadi langkah yang lebih rinci atau detail. Cara pendekatan ini sangat bermanfaat dalam pembuatan algoritma untuk masalah yang kompleks. Penghalusan langkah dengan cara memecah langkah menjadi beberapa langkah. Setiap langkah diuraikan lagi menjadi beberapa langkah yang lebih sederhana. Penghalusan langkah ini akan terus berlanjut sampai setiap langkah sudah cukup rinci dan tepat untuk dilaksanakan oleh pemroses.

3. Menulis Program
   Algoritma yang telah dibuat, diterjemahkan dalam bahasa komputer menjadi sebuah program. Perlu diperhatikan bahwa pemilihan algoritma yang salah akan menyebabkan program memiliki untuk kerja yang kurang baik. Program yang baik memiliki standar penilaian:

a. Standar teknik pemecahan masalah
   - Teknik *Top-Down*
     Teknik pemecahan masalah yang paling umum digunakan. Prinsipnya adalah suatu masalah yang kompleks dibagi-bagi ke dalam beberapa kelompok masalah yang lebih kecil. Dari masalah yang kecil tersebut dilakukan analisis. Jika dimungkinkan maka masalah tersebut akan dipilah lagi menjadi subbagian-subbagian dan setelah itu mulai disusun langkah-langkah penyelesaian yang lebih detail.
   - Teknik *Bottom-Up*
     Prinsip teknik *bottom up* adalah pemecahan masalah yang kompleks dilakukan dengan menggabungkan prosedur-prosedur yang ada menjadi satu kesatuan program sebagai penyelesaian masalah tersebut.

b. Standar penyusunan program
   - Kebenaran logika dan penulisan.
   - Waktu minimum untuk penulisan program.
   - Kecepatan maksimum eksekusi program.
   - Ekspresi penggunaan memori.
   - Kemudahan merawat dan mengembangkan program.
   - *User Friendly*.
   - *Portability.*
   - Pemrograman modular.

4. Mencari Kesalahan
   a. Kesalahan sintaks (penulisan program).
   b. Kesalahan pelaksanaan: semantik, logika, dan ketelitian.

5. Uji dan Verifikasi Program
   Pertama kali harus diuji apakah program dapat dijalankan. Apabila program tidak dapat dijalankan maka perlu diperbaiki penulisan sintaksisnya tetapi bila program dapat dijalankan, maka harus diuji dengan menggunakan data-data yang biasa yaitu data yang diharapkan oleh sistem. Contoh data ekstrem, misalnya, program menghendaki masukan jumlah data tetapi *user* mengisikan bilangan negatif. Program sebaiknya diuji menggunakan data yang relatif banyak.

6. Dokumentasi Program
   Dokumentasi program ada dua macam yaitu dokumentasi internal dan dokumentasi eksternal. Dokumentasi internal adalah dokumentasi yang dibuat di dalam program yaitu setiap kita menuliskan baris program sebaiknya diberi komentar atau keterangan supaya mempermudah kita untuk mengingat logika yang terdapat di dalam instruksi tersebut, hal ini sangat bermanfaat ketika suatu saat program tersebut akan dikembangkan. Dokumentasi eksternal adalah dokumentasi yang dilakukan dari luar program yaitu membuat *user guide* atau buku petunjuk aturan atau cara menjalankan program tersebut.

7. Pemeliharaan Program
   a. Memperbaiki kekurangan yang ditemukan kemudian.
   b. Memodifikasi, karena perubahan spesifikasi.

Pemrograman Prosedural

Algoritma berisi urutan langkah-langkah penyelesaian masalah. Ini berarti algoritma adalah proses yang prosedural. Pada program prosedural, program dibedakan antara bagian data dengan bagian instruksi. Bagian instruksi terdiri dari atas runtunan (*sequence)* instruksi yang dilaksanakan satu per satu secara berurutan oleh sebuah pemroses. Alur pelaksanaan instruksi dapat berubah karena adanya pencabangan kondisional. Data yang disimpan di dalam memori dimanipulasi oleh instruksi secara beruntun. Kita katakan bahwa tahapan pelaksanaan program mengikuti pola beruntun atau prosedural. Paradigma pemrograman seperti ini dinamakan pemrograman prosedural.

Bahasa-bahasa tingkat tinggi seperti *Cobol, Basic, Pascal, Fortran,* dan C/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* atau OOP). Paradigma pemrograman ini merupakan *trend* baru dan sangat popular akhir-akhir ini. Pada paradigma OOP, data dan instruksi dibungkus (*encapsulation)* menjadi satu. Kesatuan ini disebut kelas (*class*) dan instansiasi kelas pada saat *run-time* disebut objek (*object)*. Data di dalam objek hanya dapat diakses oleh instruksi yang ada di dalam objek itu saja.

Paradigma pemrograman yang lain adalah pemrograman fungsional, pemrograman deklaratif, dan pemrograman konkuren. Buku ini hanya

menyajikan paradigma pemrograman presedural saja. Paradigma pemrograman yang lain di luar cakupan buku ini.

Contoh Kasus dan Penyelesaian

1. Menghitung luas dan keliling lingkaran
   Proses kerjanya sebagai berikut:
   a. Baca jari-jari lingkaran
   b. Tentukan konstanta phi = 3.14
   c. Hitung luas dan keliling
      L = phi*r*r
      K = 2*phi*r
   d. Cetak luas dan keliling

2. Menghitung rata-rata tiga buah data
   a. Algoritma dengan struktur bahasa Indonesia
      - Baca bilangan a, b, dan c
      - Jumlahkan ketiga bilangan tersebut
      - Bagi jumlah tersebut dengan 3
      - Tulis hasilnya
   b. Algoritma dengan *pseudocode*
      input (a, b, c)
      Jml = a+b+c
      Rerata = Jml/3
      Output (Rerata)

3. Algoritma konversi suhu dalam derajat Celcius ke derajat Kalvin
   Penyelesaian menggunakan *pseudocode*:
   Input (Celcius)
   Kalvin = Celcius + 273
   Output (Kalvin)

1.6. Latihan

1. Buatlah sebuah rancangan program dengan menggunakan *flowchart* untuk menghitung luas lingkaran!
2. Belajar memprogram dan belajar bahasa pemrograman adalah dua hal yang berbeda. Jelaskan!

3. Di manakah letak kesalahan algoritma memutar kaset *tape recorder* di bawah ini:

   Algoritma Memutar Kaset
   1. Pastikan *tape recorder* dalam keadaan POWER ON.
   2. Tekan tombol PLAY.
   3. Masukkan kaset ke dalam *tape recorder.*