1 PENYAJIAN DATA SEDERHANA DAN STRUKTUR DATA

1.1 DATA DAN STRUKTUR DATA

Salah satu hal penting yang tidak dapat ditinggalkan dalam pemakaian komputer adalah data. Data dapat diperoleh dari berbagai sumber, seperti dari hasil pengukuran di laboratorium, hasil survi, angket, dan sebagainya.

Dengan bermacam cara, data ditransformasi menjadi informasi. Informasi sangat penting, karena memberikan dasar bagi pembuatan keputusan yang mantap dan ilmiah. Dalam suatu lembaga dan organisasi, baik yang bersifat komersial maupun industrial, bahkan organisasi yang bagaimanapun bentuknya, data dipandang sebagai suatu kekayaan yang penting dan mahal. Memang kadang-kadang data sulit diperoleh.

Komposisi data dan logika dari algoritma yang memanfaatkan data tersebut berhubungan sangat erat. Penyajian data sederhana ke dalam memori komputer akan dibahas di dalam Bab ini.

Data sederhana dapat kita himpun ke dalam suatu struktur data yang memuat informasi tentang hubungan antar item yang terdapat di dalamnya. Data sederhana yang kita miliki, terdiri dari berbagai jenis atau tipe. Dalam mengelola data yang bermacam-macam jenisnya tersebut untuk menghasilkan informasi yang baik, maka pengetahuan mengenai struktur data, amatlah penting.

Di Bab ini akan dibahas sebagai pengantar, beberapa tipe data dan bagaimana mereka dikelola oleh suatu program komputer.

Sesungguhnya tidak ada bedanya, data dan informasi bagi komputer karena komputer tidak berkepentingan, semuanya digunakan bagi kepentingan manusia pemanfaat komputer tersebut. Komputer bekerja secara elektronis yang diciptakan oleh daya pikir manusia untuk dapat membantu keterbatasan-keterbatasan yang ada pada manusia. Bekerja secara elektronis dilakukan dengan memanfaat adanya arus listrik atau tidak yang melewati rangkaian-rangkaian elektronisnya.

Bila ada arus listrik, akan dilambangkan dengan 1, bila tidak ada akan dilambangkan dengan 0. Dengan hanya berdasarkan atas dua angka yang ada tersebut (sistem bilangan binar atau biner, atau *binary*), komputer saat ini sudah dapat dikatakan sebagai alat atau barang kebutuhan primer di kota-kota besar.

Bagaimana bisa, dua angka yaitu 0 dan 1 sanggup mengolah data seperti sekarang ini (dalam jumlah yang banyak, dalam waktu yang singkat, dan dengan tingkat akurasi yang tinggi) ? Intinya, jika satu angka 0 atau 1 tersebut disebut satu **bit** (*binary digit*) yang merupakan satuan data terkecil di komputer, maka dibuat serangkaian kombinasi dari kedua angka tersebut untuk membentuk sebuah simbol (huruf, angka, tanda baca, dan sebagainya).

Ada perusahaan komputer membuat serangkaian kombinasi yang berjumlah 7 bit untuk menyimbolkan sesuatu, ada yang menggunakan 8 bit, dan sebagainya. Serangkaian bit tersebut yang menghasilkan sebuah simbol disebut dengan ***byte***.

Mulai dari *byte* (dan kumpulan dari *byte* berikutnya) dan seterusnya yang harus didefinisikan termasuk ke tipe data apa guna diolah atau diproses oleh program (program adalah serangkaian instruksi untuk komputer yang dibuat oleh *programmer*).

Sebagai contoh, kumpulan bit **1000001** disimbolkan dengan huruf **A**, dan huruf **A** tersebut termasuk dalam tipe data **karakter** atau huruf, untuk kemudian akan diproses berdasarkan (sesuai dengan) tipe datanya. Tentu saja, tidak akan diproses secara matematis karena huruf **A** bukan tipe data numerik.

Suatu struktur data adalah *suatu koleksi atau kelompok data yang dapat dikarakterisasikan oleh organisasi serta operasi yang didefinisikan terhadapnya*. Pengertiannya : struktur data adalah kumpulan elemen data (mulai dari *byte*) yang ditentukan tipe datanya, diorganisasi (dibentuk, disusun, atau dikelompokkan) dan akan diproses sesuai dengan tipe datanya.

Struktur data sangat penting dalam sistem komputer. Terhadap setiap variabel di dalam program, secara eksplisit ataupun implisit, didefinisikan struktur data yang akan menentukan operasi yang berlaku terhadap variabel tersebut.

Struktur data yang dibicarakan ini merupakan struktur data logika. Bukan penyajian secara fisik pada *storage* (memori komputer).

Pada garis besarnya, data dapat kita kategorikan menjadi :

A. Tipe data sederhana atau data sederhana, yang terdiri atas :
   a.1. Data sederhana tunggal, misalnya *integer*, *real*, *boolean*, serta karakter
   a.2. Data sederhana majemuk, misalnya *string.*
   Tipe data ini, dengan berbagai cara tertentu, dapat diorganisasikan menjadi berbagai struktur data.

B. Struktur data, meliputi :
   b.1. Struktur data sederhana, misalnya *array* dan *record*
   b.2. Struktur data majemuk, terdiri atas :
      b.2.1. *Linear*, misalnya *stack*, *queue*, serta *linear linked list.*
      b.2.2. *Nonlinear*, misalnya pohon binar (*binary tree*), pohon cari binar (*binary search tree*), pohon cari m-*way* (m-*way search tree*), *general tree*, serta *graph.*

Kedua kategori di atas, terutama diperuntukkan bagi data di dalam *storage* utama. Data yang diperuntukkan bagi *storage* tambahan, mempunyai struktur data yang dikenal sebagai organisasi *file*. Tipe organisasi *file* di antaranya adalah organisasi *sequential*, organisasi *relative,* organisasi *indexed sequential*, dan organisasi *multikey*.

Dua buah struktur data sederhana adalah *array* atau larik, dan *record*. *Array* merupakan struktur data yang terurut dan homogen, terdiri dari *item* data (kumpulan *byte* yang membentuk satu kesatuan, misalkan kumpulan huruf yang membentuk nama) yang sama tipenya. Sedangkan *record* merupakan struktur data yang boleh terdiri atas serangkaian *item* data dengan berbagai tipe data di setiap *item* datanya.

Struktur data dari tatanan yang lebih tinggi, terbentuk dari *record*, di sini termasuk daftar *linear* atau *linear list* (terutama antrean dan tumpukan), dan *graph*. Pemakaian

struktur data yang tepat di dalam proses pemrograman, akan menghasilkan algoritma yang lebih jelas dan tepat, sehingga menjadikan program secara keseluruhan lebih sederhana.

1.2 TIPE DATA SEDERHANA

Suatu jenis data tertentu akan disimpan di dalam variabel yang sesuai jenisnya. Jenis variabel menentukan rangkaian nilai yang dibutuhkan, sewaktu program dilaksanakan.

Pada umumnya, bahasa pemrograman tingkat tinggi (*high level programming language*), seperti PASCAL menginginkan agar jenis variabel yang dipakai di dalam program dinyatakan dengan jelas. Deklarasi harus digunakan untuk mengisikan satuan data ke dalam memori, serta untuk menentukan jenis operasi yang akan diterapkan terhadap data tersebut.

Beberapa bahasa pemrograman (misalnya ALGOL, PASCAL, SNOBOL) membolehkan para pemrogramnya menetapkan data-*item*nya sendiri dengan memberikan nilai yang dikirimkan kepada variabel yang bersangkutan, artinya sebuah data bukan hanya ditentukan oleh pola *binary* yang tersimpan, tetapi juga oleh jenis data yang berada di dalamnya.

Perlu ditekankan bahwa walaupun pemrogram bahasa tingkat tinggi tidak selalu perlu membuat perincian dari data yang akan disajikan untuk aplikasi pemrograman, ia perlu mengetahuinya, karena seorang pemrogram tidak hanya harus pandai menggunakan bahasa tingkat tinggi saja, namun juga harus pandai menggunakan bahasa gabungan.

Untuk menyajikan data numerik, sebagian besar bahasa pemrograman menyediakan jenis *integer* (bilangan bulat) dan jenis *real* (bilangan nyata). Selain itu, terutama untuk keperluan komputasi *scientific*, ada beberapa bahasa yang menyediakan bilangan kompleks dengan presisi ganda.

Sebagian besar bahasa pemrograman membolehkan para pemrogram memakai data karakter, dan logikal atau *boolean*. Selain itu, ada beberapa bahasa yang memakai *pointer* dan label sebagai jenis data. Intinya, berbagai macam bahasa pemrograman diciptakan orang, tentunya kita diminta untuk menggunakan bahasa pemrograman yang tepat untuk mengolah data kita. Setiap bahasa pemrograman memiliki kelebihan dan kekurangannya masing-masing, atau memiliki karakteristiknya masing-masing.

1.2.1 INTEGER

Yang dimaksud dengan *integer* adalah bilangan bulat .., -3, -2, -1, 0, 1, 2, 3, … Ia tidak mengandung pecahan dan biasanya disajikan dalam memori komputer sebagai angka bulat. Di dalam aritmetika komputer, *integer* tersebut mudah untuk disajikan dan diproses.

Besarnya nilai angka yang dapat diterima komputer adalah : -2^{N-1} hingga $2^{N-1}-1$. Di sini N merupakan jumlah bit di dalam komputer yang bersangkutan. Untuk komputer 32 bit misalnya, batasannya adalah = -2.147.483.648 hingga 2.147.483.647.

OPERASI DALAM *INTEGER*

Pada *integer* dapat dilaksanakan operasi penambahan, pengurangan, perkalian, dan pembagian *integer* (DIV), serta pemangkatan. Semua operasi di atas bekerja terhadap *inteder* (disebut *operand*), karenanya operasi disebut operasi binar. Selain itu dikenal operator *unar*, yaitu operator yang hanya mempunyai satu *operand*.

Negasi misalnya, adalah contoh dari operator *unar*.

Hasil dari pembagian *integer* DIV, adalah sebuah *integer*, yang diperoleh dengan menghilangkan bagian pecahan dari hasil pembagian. Misalnya 13 DIV 3 = 4.

Selain itu dikenal pula operasi modulo (MOD). Operasi ini hasilnya didefinisikan sebagai sisa dari pembagian. Misalnya 17 MOD 7 adalah 3. Jika hasil dari suatu proses aritmetika berada di luar jangkauan kemampuan komputer, maka akan terjadi *overflow error* dan hasil akan menjadi tidak menentu.

1.2.2 BILANGAN REAL

Data numerik yang bukan termasuk *integer*, seperti bilangan pecahan dan bilangan tak rasional, digolongkan dalam jenis data *real*. Jenis data *real* ditulis menggunakan titik (atau koma) desimal. Contohnya 3.26 43.00 -131.128, dan sebagainya.

Jika di dalam pemrograman, sebuah variabel dinyatakan sebagai bilangan *real*, tidaklah tertutup kemungkinan bahwa ia bernilai sebuah *integer* (misalnya variabel *real* Q memenuhi Q = Sqrt(X), untuk X = 9, nilai Q adalah 3, suatu nilai *real* yang sebenarnya adalah *integer*). Deklarasi suatu variabel secara berlainan, salah satu maksudnya adalah untuk menaikkan lingkup, dan ketepatan komputasi.

Bilangan *real* dimasukkan ke dalam memori komputer memakai sistem *floating point*, merupakan versi yang disebut notasi ilmiah atau *scientific notation*. Di sini penyajian terdiri atas dua bagian : *mantissa* (pecahan) dan indeks, yang disebut eksponen atau karakter yang menetapkan tempat dari titik *radix* terhadap *digit* tersebut.

Contoh 1.1

Di dalam sistem desimal : $123000 = 0.123 * 10^6$ di sini 0.123 adalah *mantissa* atau pecahan, sedangkan 6 adalah eksponennya.

$0.00151 = 0.151 * 10^{-2}$ di sini 0.151 dijadikan sebagai pecahan, dan -2 adalah eksponennya.

Jelaslah bahwa *radix* adalah 10.

Secara umum untuk bilangan *real* X, rumusnya menjadi :

$$X = M * R^E$$

di sini M dijadikan pecahan, R adalah *radix*nya dan E merupakan eksponennya.

Berhubung *radix floating point* dari suatu komputer tertentu, sudah ditetapkan (biasanya 8 atau 16), maka cukup memasukkan pecahan M dan eksponen E untuk menguraikan angka X.

Untuk menyajikan suatu bilangan secara unik, dan untuk mendapatkan banyak *digit* secara maksimum di dalam pecahan, biasanya diadakan normalisasi. Berarti bahwa pada penyajian pecahan, maka bit yang paling penting ialah suatu *digit* yang bukan 0. Normalisasi dilakukan dengan menukar tempat dari titik *radix* yang berarti menukar nilai dari eksponennya.

Variabel yang dinyatakan sebagai *real* dimasukkan ke dalam angka presisi tunggal yakni menjadi satu *string*. Bagi suatu *radix* tertentu, presisi angka ditentukan oleh penyajian *digit*nya.

Penyajian bilangan *real* ke dalam suatu komputer hanya dapat dilakukan sebagai bilangan yang pasti, oleh karena itu bilangan-bilangan *real* yang tidak pasti adalah antara dua bilangan *real* yang dapat dimasukkan ke dalam komputer. Selanjutnya, hasil dari proses tersebut harus dianggap sebagai perkiraan saja. Artinya, sulit menentukan kondisi yang tepat untuk bilangan *real*, karena di antara angka 2,0 dan 3,0 saja ada angka-angka yang berjenis *real* yang tidak terhingga banyaknya.

Jangan sekali-kali mengadakan tes dengan bentuk *statement* **if** atau **while** untuk mendapatkan persamaan yang tepat dari dua bilangan *real*; salah satu di antaranya

dihasilkan dari perhitungan aritmatika. Terutama jangan mencoba mencari persamaan hingga mencapai 0,0.

Dua kesukaran yang mungkin timbul di dalam aritmatika bilangan *real*, ialah :

1. pembatalan dua bilangan dekat tetapi tidak sama, dari tanda yang berlawanan sebagai hasil dari suatu penambahan (disebut kehilangan arti);
2. kelebihan bilangan karena proses pembagian diakibatkan oleh bilangan pembagi yangkecil.

Hindari pengetesan seperti :

X/Y < DELTA

di sini DELTA terlampau kecil. Sebaliknya pakai rumus :

X < Y * DELTA

Di dalam apliksi komersial, bilangan bulat dan pecahan kadang-kadang dinyatakan dalam bentuk *fixed-point*. Pengendalian terhadap titik *radix* diserahkan kepada pemrogram (misalnya dalam menetapkan rupiah dan sen).

1.2.3 ARITMETIKA CAMPURAN

Jika di dalam suatu bahasa pemrograman diperbolehkan memakai cara berhitung dengan aritmetika campuran, maka nilai-nilai numerik yang berbeda jenisnya dapat dicampur di dalam suatu ekspresi. Di sini *integer,* konstanta dan variabel *real* dapat digunakan bersama. Hal tersebut berarti bahwa terjadi konversi angka dari *integer* menjadi *real* dan sebaliknya.

Konversi tersebut dikerjakan secara otomatis, yakni atas perintah dari perangkat operasional yang mendukungnya seperti perangkat kompilator (*compiler*). Hanya dengan mengubah penyajian, maka konversi dari *integer* menjadi *real* akan kelihatan oleh pemrogram yang bersangkutan.

Konversi tersebut dapat dilaksanakan dengan dua cara :
1. melalui pemotongan, yakni menghilangkan pecahan dari angka *real* yang bersangkutan;
2. melalui pembulatan kepada angka *integer* yang terdekat.

Pembulatan tersebut dapat dinyatakan di dalam algoritma sebagai berikut :

```
if  x >= 0.5
  then
     begin
       x = x + 0.5;
       bulatkan x
     end
  else
     begin
       x = x – 0.5;
       bulatkan x
     end
```

Para pemrogram harus berhati-hati sekali sewaktu menggunakan sistem aritmatika campuran. Mereka harus melaksanakan konversi tersebut, atau memperhatikan bagaimana proses tersebut dilaksanakan secara otomatis.

1.2.4 BILANGAN KOMPLEKS

Suatu bilangan kompleks mempunyai bentuk umum a + bi. Di sini a dan b adalah bilangan *real*, i adalah satuan khayal.√-l (baca: akar minus satu). Jika tersedia sebagai tipe data, maka bilangan kompleks biasanya disajikan dalam dua kata yang berisi bagian *real* (yakni a), dan bagian *imaginer*nya (yakni b). Setiap bagian merupakan sebuah bilangan *real*, dinyatakan dalam *floating-point.*

1.2.5 BILANGAN PRESISI GANDA (DOUBLE PRECISION)

Di dalam beberapa bahasa pemrograman, bilangan presisi ganda tersedia dalam bentuk tipe data, dan merupakan suatu bentuk khusus dari bilangan *real*. Bilangan presisi ganda di dalam *floating-point* dinyatakan sebagai dua kata.

1.2.6 TIPE DATA BOOLEAN ATAU LOGIKAL

Suatu variabel *boolean* atau logikal hanya dapat berisikan nilai ***true*** atau ***false,*** yang kerapkali dinyatakan pula sebagai 1 dan 0. Oleh karena itu sebuah Satuan Data dapat cukup berisi satu bit saja.

Operator pada type ini berlainan dengan operator pada type integer, yaitu operator *and, or, and not.* Disini *and* dan *or adalah operator binar, sedangkan not* adalah operator

unar. Dalam urutan operasi, *not* mendapat prioritas pertama dibandingkan *and* dan *or*. Kecuali bila diberi tanda kurung.

Misalnya "A *and not* B". Di sini "*not* B" dikerjakan lebih dahulu, baru kemudian "A *and*" terhadap hasil "*not* B" tersebut.

Berikut ini tabel dari ketiga operator logikal tersebut :

Tabel 1.1 *Operator Logika*

NOT

Nilai *operand*	***Not***
true	*false*
false	*true*

AND

Nilai *Operand* 1	Nilai *Operand* 2	***And***
true	*true*	*true*
true	*false*	*false*
false	*true*	*false*
false	*false*	*false*

OR

Nilai *Operand* 1	Nilai *Operand* 2	***Or***
true	*true*	*true*
true	*false*	*true*
false	*true*	*true*
false	*false*	*false*

Nilai *true* dan *false* dapat juga dihasilkan dari apa yang dikenal sebagai operator relasional atau *relational operator* (relop). Operator relasional tidak mempunyai *operand boolean*, tetapi menghasilkan tipe *boolean*. Operator relasional tersebut adalah :

=	sama dengan
<	lebih kecil dari
>	lebih besar dari
<=	lebih kecil atau sama dengan
>=	lebih besar atau sama dengan
<>	tidak sama dengan

Sebagai contoh 6 < 12 adalah *true*
A <> A adalah *false*

1.2.7 TIPE DATA KARAKTER DAN STRING

Tipe lain dari data adalah karakter, yang elemennya merupakan aksara (simbol): (0,1,2,3,4,5,6,7,8,9,A,B,C,D,...,X,Y,Z,,?,,*...) meliputi *digit* numerik, karakter alfabetik, dan spesial karakter (simbol-simbol lain). Karakter merupakan komponen dari *string* karakter (atau disebut *string* saja), yang merupakan tipe data yang teramat penting.

Karakter yang dikenal dalam komputer biasanya mencakup :

a. 26 huruf besar dan kecil Latin (atau sesuai dengan huruf yang digunakan di suatu negara, seperti Cina, Jepang, Arab, dan sebagainya)
b. 10 digit Latin, Arab atau Romawi
c. Karakter khusus yang dapat dicetak, seperti tanda aritmatik, tanda baca, dan sebagainya
d. Ruang kosong atau *blank*
e. Karakter pengendalian (tidak dapat dicetak), seperti pengendalian kursor, dan sebagainya.

Tipe data majemuk yang terbentuk dari karakter adalah *string*. Suatu *string* adalah barisan hingga simbol yang diambil dari himpunan hingga karakter. Himpunan hingga karakter yang digunakan untuk membentuk *string* dinamakan alfabet. Sebagai contoh, himpunan *string* yang diturunkan dari alfabet (C,D,1) berisi di antaranya *string* CD1, C1D, DDC, 111D1, dan seterusnya, termasuk pula *string null* (*empty*/ hampa/ kosong). *String null* biasa dinyatakan dengan $\wedge$. Dapat dicatat bahwa $\wedge$ berbeda dengan *blank*.

String merupakan tipe data yang sangat penting dan sangat luas pemakaiannya. *String* adalah media dasar, yang dengan tipe data itulah program ditulis dan dimasukkan ke dalam komputer. *String* juga merupakan media prinsipil dalam pertukaran informasi dengan pemakai (*user*). *String* digunakan untuk menyimpan dalam *field* dari *record* pada *file*. *String* juga digunakan dalam bahasa pemrograman sebagai nama variabel, nama label serta nama prosedur.

Himpunan dari semua *string* yang mungkin dapat dibentuk dari suatu alfabet, dinamakan *vocabulary*. Suatu *vocabulary* V yang dibentuk dari alfabet A dinyatakan sebagai V_A atau A*. Jika alfabet A adalah himpunan (0,1), maka *string* yang terbentuk biasanya dikenal sebagai *bit string* (*binary digit string*).

Secara umum, marilah kita nyatakan *string* S sebagai :

$$S : a_1, a_2, ..., a_N$$

dengan setiap a merupakan anggota alfabet A.
Panjang dari *string*, didefinisikan sebagai banyaknya karakter pembentuk *string* tersebut, misalnya sebanyak N karakter. Kita tulis :

$$|S| = N \quad \text{atau} \quad \text{length}(S) = N$$

Di sini panjang *string null*, $|\wedge|$ adalah 0, sedangkan panjang dari *blank* adalah 1.

Perlu diketahui bahwa operasi yang berlaku terhadap *string* dapat disebutkan adalah :

* **LEN(*string*)**, yakni operasi untuk menghitung *length* atau panjang *string*.
* **CONCATE(S_1,S_2)**, yakni *concatenation* (produk/penyambungan 2 buah *string*.)
* **SUBSTRING**, yakni operasi mengambil *substring* (bagian) dari suatu *string*.

Hasil dari operasi *length* adalah suatu tipe data *integer*. Sedangkan hasil dari *concatenation* maupun *substring* adalah memiliki tipe data *string* juga.

Jika S_1 adalah string : $a_1, a_2, ..., a_N$
S_2 adalah string : $b_1, b_2, ..., b_N$
maka CONCATE(S_1,S_2) = $a_1, a_2, ..., a_N, b_1, b_2, ..., b_N$

Contoh lain :
nam1 = ‘Adhini’
nam2 = ‘Novia’
hsl = CONCATE(nam1,nam2)
maka, nilai hsl = ‘AdhiniNovia’

Substring adalah operasi untuk membuat suatu *string* baru dengan cara mengambil beberapa karakter berurutan dari suatu *string* yang diketahui.
Bentuk umum dari operator *substring* adalah :

SUBSTRING(S, i, j)

di sini S adalah *string* semula
i adalah posisi karakter awal yang diambil
j adalah banyaknya karakter yang diambil
i dan j bertipe *integer*.

Sebagai contoh :
Diketahui S adalah *string* $a_1, a_2, a_3, ..., a_N$,
maka SUBSTRING(S,2,3) adalah a_2, a_3, a_4

Contoh lain :
npm = '10102563'
thm = SUBSTRING(npm,4,2)
maka, nilai thm = '02'

Perhatikan bahwa ketiga operator *length*, *concatenation* serta *substring* tersebut tidak berlaku bagi data yang bertipe integer. Namun, kadang-kadang untuk suatu keperluan tertentu, data *integer* dipindahkan menjadi *string*, lalu dimanipulasi sebagai *string*. Jadi misalnya 12 + 35 = 47 dalam *integer*, apabila mereka dimanipulasi secara *string* (penambahan = penggabungan = *concatenation*), maka "12" + "35" = "1235" atau CONCATE("12", "35") = "1235"

Di bahasa pemrograman BASIC misalnya, *concatenation* dilambangkan dengan tanda plus (+). Di BASIC, bisa juga nilai *integer* yang sudah dipindahkan menjadi *string* tersebut dimanipulasi secara *integer*. Misalkan, VAL("12") + VAL("35") = 47.

Selain operator di atas, masih ada operator lain yang berlaku pada *string*, yakni operator INSERT. Pada operasi INSERT ini dibutuhkan dua *operand string* dan sebuah *operand integer*. Hasilnya berupa data yang bertipe *string*.

Bentuk umumnya adalah :

$$\boldsymbol{INSERT}(S_1,j,S_2)$$

Ini berarti kita menyisipkan *string* S_2 di dalam *string* S_1 sedemikian sehingga karakter pertama dari S_2 menggantikan posisi karakter ke j dari S_1 hingga selesai dengan seluruh *string* S_2 dan dilanjutkan lagi dengan sisa *string* S_1.

Contohnya:

S1 adalah *string* : $a_1, a_2, a_3, ..., a_N$

S2 adalah *string*: $b_1, b_2, b_3, ..., b_N$

maka INSERT(S1,3,S2) adalah $a_1, a_2, b_1, b_2, b_3, ..., b_N, a_3, ..., a_N$

Contoh lain :

x = 'Ella Mia'
y = 'Nova '
z = INSERT(x,6,y)
maka, z = 'Ella Nova Mia'

Selain itu masih terdapat operator DELETE. Operator ini mengandung sebuah *operand string* dan 2 *operand integer*.

Bentuk umumnya adalah :

DELETE*(S,i,j)*

Ini berarti kita menghapus *substring* yang panjangnya j, bermula pada posisi ke i, dari S.

Sebagai contoh :

S adalah : a_1, a_2, a_3,..., a_N

Maka, DELETE(S,3,2) hasilnya : a_1, a_2, a_5, a_6..., a_N

Contoh lain :

x = ‘Heru Purnomo’

y = DELETE(x,3,1)

maka, nilai y = ‘Heru Purnomo’

Dalam bahasa pemrograman, *string* dijelaskan dengan pemberian tanda “ “ atau ‘‘. Jadi data 43 adalah *integer*, sedangkan data "43" adalah *string*. Juga pada beberapa bahasa pemrograman, seperti BASIC, variabel *string* diberi ciri pada nama variabelnya, yakni dengan tambahan $, misalnya NAMA$, dan sebagainya.

Operator lainnya adalah INDEX. Operator ini ditujukan untuk mendapatkan posisi ke berapa suatu substring di dalam suatu string.

Bentuk umumnya :

INDEX*(string, substring)*

Contoh : a = ‘Rin Novia’

b = ‘No’

c = INDEX(a, b)

maka, nilai c = 5 (yang bertipe *integer*).

Ada bahasa pemrograman yang menyediakan fasilitas operasi ini dengan *statement* AT atau POS.

Operator yang lain adalah REPLACE. Operator ini ditujukan untuk mengganti suatu *substring* di dalam suatu *string*.

Bentuk umumnya :

REPLACE*(teks, string_lama, string_baru)*

Contoh : REPLACE(‘Enny Prima’, ‘ny’, ‘ni’) = ‘Enni Prima’
atau
s = ‘Enny Prima’
t = ‘ny’
u = ‘ni’
v = REPLACE(s, t, u)

maka, nilai v = ‘Enni Prima’

1.3 DEKLARASI DATA DALAM BAHASA PEMROGRAMAN

Bahasa pemrograman mengharuskan pemrogram menetapkan tipe data bagi variabel yang digunakan. Semua operasi yang dilakukan terhadap variabel tersebut harus sesuai dengan operasi yang berlaku pada tipe yang bersangkutan. Masing-masing bahasa pemrograman mempunyai cara sendiri-sendiri dalam pemberian tipe data kepada variabel. Ada kompilator yang sangat serius dalam menanggapi masalah ini, ada pula yang tidak.

Misalnya waktu memasukkan *string* "Q", kita lupa memberi tanda petik, ada kompilator yang membetulkan sendiri, dan meneruskan program tanpa menyalahkan kita. Namun ada pula kompilator yang meminta kita memperbaikinya, sebelum program diteruskan. Bahasa PASCAL dan COBOL mengharuskan kita secara eksplisit/ nyata mendeklarasikan tipe dari setiap variabel yang kita gunakan dalam program. Bahasa lain,. Misalnya FORTRAN, menyatakannya secara tidak eksplisit. Di sini variabel dengan nama dimulai dengan huruf I, J, K, L, M atau N adalah tipe *integer*. Dalam COBOL misalnya, terdapat DIVISION khusus untuk menyatakan tipe data. Sedang pada bahasa lain, mungkin cukup pada saat diperlukan saja.
PL/1 mempunyai *statement* DECLARE, PASCAL mempunyai *statement* var. Sedangkan FORTRAN mempunyai *statement* INTEGER, REAL serta DIMENSION.

Berikut ini diperlihatkan bagaimana mendefinisikan variabel bertipe *integer*, *boolean* dan karakter dalam COBOL dan PASCAL. Di sini variabel *integer* kita namakan HITUNG, mempunyai nilai maksimum 3 digit; variabel *boolean* kita namakan PANJI, dan variabel karakter kita namakan BETA.

COBOL :

```
DATA DIVISION.
01 HITUNG PIC S999.
01 FLDA PIC X.
     88 PANJI VALUE ''Y''.
01 BETA PIC X.
```

Huruf S dalam PICTURE dari HITUNG menyatakan tanda atau *sign*. Ini berarti diperbolehkan adanya nilai negatif dan positif untuk disimpan sebagai harga HITUNG. Bahasa COBOL tak mempunyai tipe data *boolean*, namun kita dapat menggunakan *condition-name* yang menyatakan *true* dan *false.*

Di sini misalnya, *level* 88, *condition-name* PANJI didefinisikan mempunyai nilai *boolean true,* bila FLDA bernilai "Y". Untuk itu, pemrogram dalam prosedur dapat menulis :

```
IF PANJI THEN...
     ELSE...
```

yang mempunyai arti sama dengan :

```
IF FLDA = ''Y''  THEN ……
     ELSE...
```

PICTURE X pada FLDA serta BETA menunjukkan tipe karakter, sedangkan PICTURE 9 pada HITUNG menunjukkan tipe numerik.

Dalam Bahasa PASCAL, hal di atas dapat kita tulis :

```
var HITUNG  : integer;
    PANJI   : boolean;
    BETA    : char;
```

Banyaknya *digit* dalam HITUNG dinyatakan dalam format *statement* **read** dan **write**. Terhadap variabel PANJI dapat diberikan nilai *false* atau *true.*

String dapat dengan mudah dideklarasikan dalam COBOL. Misalnya, variabel ALAMAT dengan panjang *string* 25 karakter, dideklarasikan sebagai

```
01 ALAMAT PIC X(25).
```

Sedangkan dalam PASCAL sebagai :

```
var alamat packed array[l..25] of char;
```

Pascal memperkenankan pemrogram untuk menggunakan *statement* **type** guna menyatakan suatu struktur data majemuk. Kemudian, variabel yang mempunyai bentuk serupa, dinyatakan dengan mudah. Contohnya, suatu struktur data majemuk yang

terbentuk dari tipe elementer *char*, sebanyak 25 karakter, dan kita beri nama *string*25, kita nyatakan sebagai :

```
type String25 : packed array[l..25] of char;
```

Variabel NAMA dan ALAMAT yang bentuknya sama, kita nyatakan sebagai :

```
var NAMA, ALAMAT : String25
```

Beberapa Bahasa mempunyai operator siap pakai dalam manipulasi variabel *string*. PL/1 mempunyai LENGTH, SUBSTR, dan // (untuk *concatenation*), COBOL mempunyai STRING dan UNSTRING. Sedangkan Pascal tidak mempunyai fasilitas tersebut, Pemrogram harus membuat *routine* untuk melaksanakan operasi *string* tersebut.

1.4 PEMETAAN (*MAPPING*) KE *STORAGE* : *INTEGER*

Struktur data secara logik dapat mempunyai beberapa *storage mapping* atau representasi fisik. Hal ini merupakan salah satu cirinya. Salah satu *storage mapping* yang dapat dilakukan terhadap *integer* adalah apa yang disebut bentuk *sign-and-magnitude. Integer* positif didefinisikan dengan tanda plus serta sebarisan digit yang menyatakan *magnitude*/ besamya; sedangkan *integer* negatif didefinisikan dengan tanda minus serta sebarisan *digit*. Tanda plus kerap kali diabaikan penulisannya dalam penggunaan sehari-hari. Namun, tanda plus tersebut harus dinyatakan ketika melakukan pemrosesan bilangan dengan komputer.

Juga, kalau sehari-hari bilangan dinyatakan dalam basis 10 (sistem desimal), maka dalam memori komputer kita menyatakan bilangan dalam basis 2 (sistem *binary*). Manusia dengan mudah bekerja terhadap bilangan dalam bentuk *sign-and-magni-tude* tersebut. Namun, apabila kita akan melakukan penjumlahan dengan komputer, dengan kedua *operand* berbeda tanda, penjumlahan akan beralih menjadi pengurangan yang kadang-kadang menimbulkan kesukaran. Untuk itu, kita gunakan apa yang disebut *complement*, yang dapat diterangkan sebagai berikut :

Diberikan 3 *integer* non negatif X, X’ dan R. Kita definisikan X’ adalah *complement* dari X terhadap R (atau R's *complement* dari X) bila X + X' = R.

Dalam penyajian *complement*, X disebut bentuk "true” dan X’ disebut bentuk *complement*. Dalam komputer *binary*, R dipilih merupakan pangkat dari 2.

$$R = 2^N$$

Pemilihan N menentukan daerah rentangan dari integer yang dapat disajikan.

Mapping dengan $R = 2^N$ dikenal sebagai sistem *two's complement*. Tabel berikut menunjukkan *binary sign-and-magnitude*, dan *two's complement* (menggunakan N = 4) dari integer -7 sampai 7.

Tabel 1.2 *Sign and Magnitude*

Integer	*Binary sign-and-magnitude*	*Two's complement*
-7	-111	1001
-6	-110	1010
-5	-101	1011
-4	-100	1100
-3	-011	1101
-2	-010	1110
-1	-001	1111
+0	+000	0000
+1	+001	0000
+2	+010	0001
+3	+011	0010
+4	+100	0011
+5	+101	0100
+6	+110	0101
+7	+111	0110

Bentuk *complement* X' = R–X menyatakan *integer* negatif X dan bentuk "true" menyatakan *integer* positif X.

Algoritma untuk melakukan perhitungan aritmatika terhadap *integer* yang dinyatakan dalam bentuk *complement* adalah lebih mudah bagi komputer dibandingkan dalam bentuk *sign-and-magnitude*.

Jadi, apabila banyak komputasi dilakukan terhadap *integer*, dianjurkan untuk menyatakan *integer* dalam bentuk *complement*.

Pilihan lain untuk bentuk *complement* adalah mengambil konstanta komplemen R = 2^N-1, yang disebut sistem *one'-s complement*. Sama seperti *two's complement*, *one's complement* mempunyai kelebihan dalam pelaksanaan perhitungan aritmatika dibandingkan dengan sistem *sign-and-magnitude*. Sedangkan antara *one's* dan *two's complement* masing-masing mempunyai kelebihan dan kekurangan.

1.5 PEMETAAN KE *STORAGE* : KARAKTER DAN *STRING*

Banyak aturan yang dapat kita gunakan untuk menyatakan tipe data karakter dalam *storage*. Dua di antaranya sangat terkenal, yakni *Extended Binary Coded Decimal Interchange Code* (EBCDIC) dan *American Standard Code for Information Interchange*(ASCII).

EBCDIC, yang dikembangkan oleh IBM, adalah kode 8 bit. Di sini dibutuhkan 8 *binary digit* (bit) untuk menyatakan satu karakter dalam alfabet: Dalam 8 bit terdapat 2^8 (=256) kemungkinan. Nyata bahwa pada EBCDIC, cukup banyak macam karakter yang dapat digunakan. Di sini meliputi huruf besar serta kecil, *digit* numerik dan banyak karakter khusus.

ASCII adalah cara pengkodean 7 bit. Terdapat 2^7 (=128) kemungkinan, separuh dari yang dimiliki EBCDIC. Penggunaan ASCII ini disebabkan karena lebih sedikitnya *storage* yang dipakai, serta lalu lintas data karakter tersebut dapat lebih cepat.

Selain kedua cara pengkodean tersebut di atas, masih terdapat banyak cara lagi, di antaranya adalah cara BCD (*Binary Coded Decimal*) yang menggunakan 4 bit setiap karakternya.

Selain itu, di antara sekian banyak cara *mapping* ke *storage* terhadap karakter tersebut, ada suatu cara yang sengaja diciptakan untuk aplikasi khusus, seperti kode Huffman. Di sini, Karakter disajikan oleh bit yang banyaknya variabel, tergantung pada frekuensi relatif kemunculan karakter tersebut dalam *vocabulary* dari aplikasi. Karakter yang sering muncul dinyatakan dalam pola bit yang lebih pendek, sedangkan karakter yang jarang muncul, dinyatakan dalam pola bit yang lebih panjang.

Sebagai contoh dalam aplikasi khusus, karakter 0, dinyatakan dalam *single bit* 0, karakter A dinyatakan dalam 5 *bit* 10101, sedangkan karakter % (sangat jarang muncul) dinyatakan dalam 16 *bit* 1011111111111001. Apabila dihitung, rata-rata penggunaan *bit* adalah 2.91 bit/ karakter.

Banyak komputer memperkenankan pemrogram menggunakan berbagai kode pada sebuah program. Sebagai contoh, pada COBOL, data dapat dibaca dan ditulis ke *file* menggunakan EBCDIC, pada *tape drive* misalnya. Data lain dapat dibaca/ ditulis dengan ASCII, misalnya pada peralatan *input/output terminal*. Data lain mungkin dalam kode yang lain pula.

Apabila suatu *mapping karakter* digunakan untuk menyatakan *integer*, tanda plus atau minus harus ikut disimpan. Secara umum, tanda ini disimpan di sebelah kanan *digit* dari bilangan.

Banyak kompilator menawarkan pilihan lain untuk *mapping integer*. Yang paling luas digunakan adalah bentuk *packed-decimal*. Di sini data numerik disimpan dengan menggunakan 2 *digit* setiap 8 *bit*, tidak seperti EBCDIC 1 *digit* setiap 8 *bit*. Pada 8 *bit* terakhir disimpan selain digit derajat terendah, juga tanda dari bilangan tersebut.

Berikut ini perbandingan kode EBCDIC, ASCII, dan *packed-decimal* untuk menyatakan +903 dan -903.

Tabel 1.3 *Perbandingan Kode EBCDIC, ASCII, dan Packed-Decimal*

	+ 903			
EBCDIC ASCII PACKED DECIMAL	11111001 0111001 10010000	11110000 0110000 00111100	11110011 0110011	01001110 0101011
	-903			
E B C D I C ASCII PACKED DECIMAL	11111001 0111001 10010000	11110000 0110000 00111100	11110011 0110011	00100000 0101101

1.6 SEKILAS MACAM STRUKTUR DATA

Algoritma bekerja atas dasar data yang menunjukkan fakta dari permasalahan kita dalam dunia nyata. Algoritma dan data menentukan hasil dari pelaksanaan program, dan mempunyai hubungan simbolik. Dalam hal ini, semakin cocok komposisi data untuk

suatu-aplikasi tertentu, semakin mudah algoritma tersebut melaksanakan tugasnya. Untuk menyederhanakan pelaksanaan algoritma, dianjurkan untuk mengatur data menjadi suatu unsur logikal yang lebih tinggi dibandingkan variabel biasa. Unsur logikal tersebut membentuk struktur data.

Dengan perkataan lain, suatu struktur data merupakan koleksi dari satuan data sederhana yang terorganisir dengan aturan tertentu. Oleh karena itu, suatu struktur data dicirikan oleh :

1. jenis atau tipe satuan data pembentuknya
2. hubungan antara satuan data tersebut.

Contoh 1.2

Sebuah program akan dirancang untuk memberi simulasi kepada kemampuan kerja komputer. Algoritma simulasi ini, memerlukan data tentang waktu dari pekerjaan yang akan diproses. Sejumlah satuan data dibentuk, masing-masing menunjukkan waktu tiba dari pekerjaan, kebutuhan data oleh komputer, dan sebagainya. Kemudian program akan mencontoh proses dari pekerjaan semu yang dibentuk oleh sistem komputer, untuk disimulasikan. Program akan menjadi lebih jelas dan lebih sederhana apabila data yang menjadi bagian dari tiap-tiap pekerjaan, diorganisir ke dalam suatu unsur tunggal, yakni berupa sebuah *record*.

Kemudian, akan menjadi jelas lagi apabila *record* seperti itu diatur membentuk suatu struktur data yang lebih tinggi lagi tingkatannya, sehingga menjadi bentuk *first-in-first-out* (FIFO), sesuai dengan waktu tiba dari pekerjaan yang diwakilinya. Struktur data seperti itu disebut antrean atau *queue*.

Struktur data tertentu, seperti *array* atau larik biasanya selalu tersedia bagi pemrogram di dalam bahasa pemrograman tingkat tinggi yang banyak dipakai. Namun ada pula struktur yang tidak tersedia, seperti misalnya struktur *stack* atau tumpukan. Untuk itu, harus disusun sendiri oleh Pemrogram dengan menggunakan variabel bertipe sederhana, ataupun dari struktur yang telah tersedia.

Sebuah *array* (satu atau beberapa dimensi) merupakan dasar dari struktur data. Sebuah *array* dapat menyatukan satuan data sederhana, dan sejenis, yang masing-masingnya mendapat nama secara kolektif (nama *array* yang bersangkutan), dan dari indeks atau subskrip (*subscript*) yang memberikan identifikasi terhadap posisi mereka di dalam *array* tersebut. *Array* dari *string* juga penting.

Sebuah *Record* terbentuk dari beberapa tipe data sederhana dan termasuk *string*. Data tersebut dibentuk sebagai unsur tunggal karena mereka memberikan gambaran dari obyek

permasalahan yang nyata sehari-hari. *Record* biasanya ditingkatkan menjadi struktur data yang lebih tinggi, melalui nilai dari salah satu satuan datanya (disebut *field*).

Variabel yang sederhana, *record* dan juga *array*, seperti kita katakan terdahulu, dapat diorganisir menjadi struktur data yang lebih tinggi. Struktur tersebut banyak yang bersifat dinamis, artinya bahwa dalam pelaksanaan program yang memakai struktur data tersebut, banyaknya komponen dapat berubah, ada komponen yang dapat dihilangkan dan ada yang dimasukkan.

Contoh struktur data yang lebih tinggi tersebut adalah daftar *linear* (seperti tumpukan dan antrean), pohon binar dan graf (*graph*). Struktur tersebut dimanfaatkan untuk mengorganisir data yang berada di dalam memori utama.

1.7 ORGANISASI LOGIK DAN FISIK DARI STRUKTUR DATA

Memori komputer dapat kita bayangkan terdiri atas barisan atau untai sel-sel (masing-masing sel berisi satu *binary digit* atau *bit*) dengan alamatnya sekaligus. Setiap untai dirangkai dari sejumlah bit yang ditentukan jumlahnya dalam satu untai oleh model komputer tertentu.

Struktur data terdiri dari satuan data sederhana yang cocok untuk program yang memakainya. Hubungan antara satuan data tersebut membentuk salah satu ciri dari struktur yang bersangkutan. Jika sebuah struktur data langsung tersedia dalam bahasa pemrograman (misalnya *array* terdapat di dalam FORTRAN), maka struktur data tersebut langsung dapat dipakai. Jika struktur tersebut tidak tersedia, maka pemrogram harus membuatnya terlebih dahulu dari tipe data yang tersedia. Berhubung struktur data tersebut harus dimasukkan ke dalam memori berupa untai *bit*, maka setiap struktur diberi ciri oleh organisasi logikal (perlu ada hubungan antar komponen sewaktu diaplikasikan), dan oleh organisasi fisikal (penempatan dalam memori).

Jika satuan data sederhana dapat membentuk sebuah struktur yang lebih hemat dalam memori, maka struktur data tersebut disatukan: beberapa satuan data ditempatkan sebagai satu untai. Struktur tersebut tidak dapat langsung ditujukan kepada sebuah alamat (yang bisa hanyalah sebuah untai atau *byte*); untuk itu haruslah diusahakan melalui proses pemrograman.

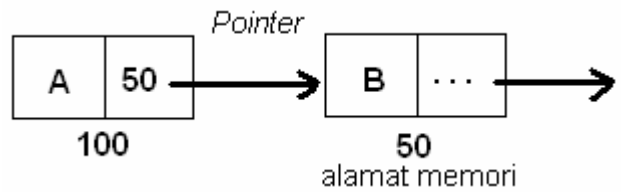
Jika menggunakan penyajian secara sekuensial, maka komponen struktur data ditempatkan ke dalam lokasi memori secara berurutan. Hubungan antara komponen-komponen tersebut tidak nampak dari proses mereka.

Contoh 1.3

Sebuah himpunan data logikal (masing-masing sepanjang 1 *bit*) dapat dijadikan untai 16 bit memakai 16 elemen himpunan ke dalam satu untai. Dua cara utama untuk menempatkan memori terdapat di dalam struktur tingkat tinggi seperti daftar.

Cara lain adalah penyajian secara berkait, atau *linked*. Di sini setiap komponen dari struktur data dilengkapi dengan satu atau beberapa satuan data tambahan, yang dipergunakan untuk melakukan hubungannya dengan komponen yang ada di sekelilingnya. Satuan data tambahan seperti itu disebut *pointer* atau *link* atau penuding yang berisikan alamat memori dari satuan data lain di dalam struktur data.

Dengan adanya *pointer* tersebut, akan terjalin relasi antar komponen dalam struktur data tersebut. Di dalam gambar 1-1, terlihat bahwa komponen A menuding komponen B.



Gambar 1.1. *Penuding (pointer)*

Di sini berarti bahwa nilai yang disimpan di dalam lokasi *pointer* adalah 50, sesuai dengan alamat dari data pertama komponen B. *Pointer* (*link*) berfungsi menetapkan bagaimana organisasi dari struktur data yang bersangkutan.

Penyajian data secara sekuensial bersifat sangat kaku. Hal ini disebabkan karena untuk menghapus komponen dari struktur dan menyisipkan komponen ke dalamnya, hanya dapat dilakukan melalui reorganisasi fisik dari struktur di dalam memori. Oleh karena itu, ia baru dapat diaplikasikan dengan baik apabila komposisi dari struktur tersebut tidak banyak berubah selama pelaksanaan program yang bersangkutan.

Penyajian dengan cara berkait (*linked*) bersifat lebih fleksibel, karena untuk mengadakan perubahan struktural, kita cukup melakukan perubahan beberapa *pointer* saja. Oleh sebab itu, penyajian berkait ini cocok bagi struktur data yang dinamis. Tingkat

keluwesan tersebut terdapat dalam ruang memori dan waktu yang dibutuhkan, guna mengarahkan *pointer* yang bersangkutan, yang juga merupakan elemen struktural.

Jika kita memakai sistem penyajian berkait, maka jumlah memori yang tersedia bagi struktur data, dibagi menjadi dua bagian. Satu bagian menampung struktur yang ada tersebut, sedangkan bagian yang lain (disebut ruang yang tersedia), merupakan tempat cadangan, untuk menampung komponen yang masih "kosong." Jika diperlukan sebuah ruang untuk menempatkan komponen baru, maka akan diambilkan dari ruang cadangan tersebut.

Jika suatu komponen tidak dibutuhkan lagi, maka ruang dikembalikan kepada tempat cadangan yang bersangkutan. Ada kalanya juga terjadi bahwa komponen yang tidak dibutuhkan lagi dikumpulkan secara periodik untuk dikembalikan ke tempat cadangan.

1.8 WAKTU PELAKSANAAN PROGRAM, FUNGSI UKURAN INPUT

Dalam menyelesaikan suatu masalah menggunakan komputer, kerap kali kita memiliki banyak pilihan algoritma yang dapat kita gunakan. Di sini kita harus memilih salah satu algoritma yang terbaik untuk kita gunakan menyelesaikan masalah tersebut. Dalam hal ini, berdasarkan hal apakah, pilihan terbaik kita tersebut ?

Ada dua tujuan, yang kadang-kadang saling bertentangan, dan tidak dapat dicapai bersama. Tujuan pertama, kita menginginkan algoritma tersebut mudah untuk di*debug* atau diperbaiki. Atau kedua, kita menginginkan algoritma yang mengefisienkan pemakaian komputer kita, terutama dalam penggunaan waktu. Di sini diharapkan penggunaan waktu yang sesingkat-singkatnya bagi pelaksanaan program.

Kalau kita menulis program yang hanya digunakan satu atau beberapa kali saja, tujuan pertama di atas lebih diutamakan. Kita meminimalkan biaya pembuatan program, karena biaya tersebut sangat jauh di atas biaya komputer untuk pelaksanaan program.

Sebaliknya, apabila program digunakan banyak kali, biaya pemakaian komputer jauh di atas biaya pembuatan program. Karena itu tujuan kedua yang kita utamakan. Kita pilih algoritma yang menggunakan waktu pelaksanaan minimal, meskipun algoritma itu rumit dan sulit dimengerti.

Waktu pelaksanaan program tcrgantung pada beberapa faktor, di antaranya input yang diberikan pada program, mutu dari kode pada kompilator untuk menghasilkan program obyek, keadaan serta kecepatan instruksi pada mesin untuk melaksanakan program dan waktu kompleksitas algoritma yang mendasari program tersebut.

Kenyataan bahwa waktu pelaksaan program tergantung pada input, membawa kita untuk mendefinisikan waktu pelaksanaan tersebut sebagai suatu fungsi input. Ia terutama tergantung kepada banyaknya input. Suatu contoh yang baik adalah proses pengurutan atau *sorting*. Dalam problema *sorting*, sebagai input adalah sebuah daftar bilangan atau kata yang akan diurutkan. Sebagai *output* adalah daftar yang sudah terurut. Contohnya, diberikan bilangan 2,1,3,1,5,8 sebagai input. Setelah dilakukan *sorting* diperoleh daftar 1,1,2,3,5,8 sebagai *output*. Ukuran input dalam problema *sorting* ini adalah banyaknya bilangan. Waktu pelaksanaan program pengurutan ini tergantung kepada banyaknya data atau panjangnya input, kecuali dalam hal-hal khusus.

Kita menulis T(n) sebagai fungsi waktu pelaksanaan program dengan ukuran input n. Contohnya, suatu program mempunyai fungsi pelaksanaan $T(n)= cn^2$, c adalah suatu konstanta. Satuan dari T(n) tidak kita spesifikasikan, namun kita boleh berpikir bahwa T(n) adalah banyaknya instruksi yang dilaksanakan pada komputer ideal.

Untuk beberapa program tertentu, waktu pelaksanaan selain tergantung pada ukuran input, juga tergantung pada bentuk khusus dari input. Dalam hal ini, kita mendefinisikan T(n) sebagai *worst case running time*, yakni waktu pelaksanaan maksimum yang diperlukan, untuk input yang bagaimanapun bentuknya. Juga dikenal *best case running time*.

Seperti telah disebutkan di atas, waktu pelaksanaan program tergantung pula pada kompilator yang digunakan untuk melaksanakan program tersebut. Hal inilah yang mengakibatkan kita tidak dapat mendefinisikan waktu pelaksanaan algoritma itu sebanding dengan n^2, atau laju kenaikan algoritma itu adalah n^2. Konstanta pembanding belum dapat dispesifikasi. Ia banyak tergantung kepada kompilator, mesin serta faktor lain.

1.9 NOTASI BIG-OH

Dalam analisis laju kenaikan fungsi waktu pelaksanaan, kita biasa menggunakan apa yang disebut notasi big-oh. Contohnya, kita katakan waktu pelaksanaan T(n) suatu program adalah $0(n^2)$, dibaca big-oh n kuadrat atau oh n kuadrat. Di sini diartikan bahwa ada konstanta positif c dan n_0 sedemikian sehingga untuk setiap $n >= n_0$. T(n) lebih kecil atau sama dengan cn^2. Sebagai contoh lain, misalkan T(n) = 2n. Fungsi T(n) ini dapat termasuk

dalam keluarga O(g), untuk g(n) 3^n. Di sini selalu berlaku $2^n < 3^n$ untuk semua n > 0. Dalam definisi yang baru lalu, dapat dimasukkan konstanta c = 1 dan n_0= 0.

Hal sebaliknya, yakni g(n) termasuk keluarga O(T) adalah tidak benar. Hal ini dapat dibuktikan dengan pembuktian tidak langsung. Kita andaikan g(n) termasuk dalam O(T). Karena itu terdapat konstanta positif c dan n_0 sedemikian sehingga untuk setiap n >= n_0 berlaku :

$$3^n <= c.2^n$$

Hubungan ini berakibat :

$$n \log 3 <= \log c + n \log 2$$
$$n(\log 3 - \log 2) <= \log c$$
$$n \log 3/2 <= \log c$$
$$n <= \frac{\log c}{\log(3/2)}$$

suatu kontradiksi.

Untuk lebih jelas lagi, kita ambil satu contoh lain. Fungsi $T(n) = 3n^3 + 2n^2$ adalah $O(n^3)$. Untuk melihat hal ini, misalkan n_o = 0 dan c = 5. Dapat kita tunjukkan bahwa untuk n >= 0 selalu $3n^3 + 2n^2 <= 5n^3$.

Jika kita mengatakan bahwa T(n) adalah 0(f(n)), maka f(n) merupakan batas atas dari laju kenaikan T(n). Sebaliknya kita dapat pula menyebutkan batas bawah dari laju kenaikan T(n), dengan menyebutnya sebagai "T(n) adalah W(g(n)), big-omega dari g(n)". Ini berarti, bahwa ada konstanta c sedemikian sehingga T(n) > cg(n), untuk sejumlah tak hingga nilai n. Sebagai contoh adalah T(n) $3n^3 + 2n^2$ yang merupakan $W(n^3)$. Di sini, misalkan c = 1, berlaku $T(n) > cn^3$, untuk n = 0,1, . . .

Sebagai contoh lain, pandang :

$$T(n) \begin{cases} n & \text{untuk } n >= 1,\ n \text{ ganjil} \\ n/100 & \text{untuk } n >= 0,\ n \text{ genap} \end{cases}$$

T(n) ini adalah $W(n^2)$. Sebagai c, kita ambil c = 1/100. Untuk n = 0, 2, 4, 6, ... genap, tak hingga, selalu berlaku T(n) > n/100.

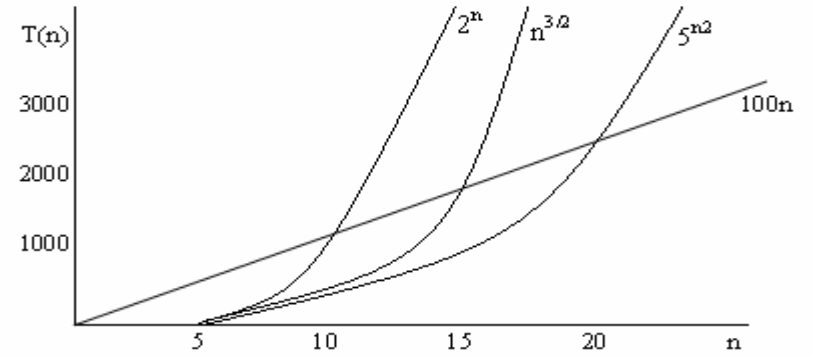
Untuk melakukan evaluasi program, kita dapat melakukannya dengan memperbandingkan fungsi waktu pelaksanaan, tanpa kita perhatikan konstanta perbandingan c. Dengan cara seperti itu, kita hanya melihat pada notasi big-oh- nya. Jadi,

sebagai contoh program dengan waktu $0(n^2)$ adalah lebih cepat dibandingkan program dengan waktu $O(n^3)$. Tentunya evaluasi ini dilakukan dengan memandang bahwa kita menggunakan kombinasi kompilator-mesin yang sama. Mungkin timbul pertanyaan, apakah pengabaian konstanta perbandingan c tersebut benar-benar meyakinkan hasil evaluasi kita ?, mengingat misalnya ada sebuah program membutuhkan waktu hanya $5n^3$ mili detik. Benarkah $O(n^3)$ lebih cepat dari $O(2^n)$? Memang untuk n kecil, yakni untuk n < 20, program pertama lebih lamban dari program kedua. Tetapi begitu nilai n meningkat besar, perbandingan waktu pelaksanaan kedua program tersebut naik pesat. Bayangkan untuk n yang lebih besar, perbandingan waktu pelaksanaan kedua program tersebut naik pesat.

Bayangkan untuk n yang lebih besar sekali pada perbandingan $5n^3/100n^2 = n/20$ tentunya kita semua mengetahui bahwa masalah kita justru timbul untuk ukuran input yang besar. Jadi kita tak salah, bila mengatakan program dengan waktu $O(n^2)$ adalah lebih baik dari program dengan waktu $O(n^3)$. Lebih-lebih bila program memang didesain untuk menyelesaikan masalah dengan ukuran input yang besar.

Satu alasan lain untuk mengevaluasi laju kenaikan waktu pelaksanaan program adalah diperlukannya hal tersebut untuk mengukur kemampuan komputer yang kita gunakan menjalankan program tersebut.

Kalau dihitung, ternyata beban komputer tersebut sangat berat, artinya waktu yang diperlukannya untuk menjalan program tersebut sangat lama. Dalam hal ini, kita dapat mengganti komputer tersebut dengan komputer lain yang lebih cepat. Tentunya dapat pula dikombinasikan dengan penggantian kompilator. Penggantian kombinasi kompilator-mesin tersebut tidak terlalu efektif lagi bagi program dengan laju kenaikan waktu yang besar. Bagi program dengan laju kenaikan waktu yang rendah, seperti program dengan waktu O(n), serta 0(n log n) penggantian di atas sungguh efektif.



Gambar 1.2. *Grafik fungsi waktu*

Sebagai contoh, perhatikan Gambar 1-2 yang menunjukkan grafik fungsi waktu pelaksanaan program dari 4 buah program yang berbeda waktu pelaksanaannya. Waktu pelaksanaannya diukur dalam detik, dihitung pada pemakaian kombinasi kompilator-mesin tertentu yang sama untuk keempat program tersebut. Misalnya kita mempunyai waktu 1000 detik atau lebih kurang 17 menit untuk menyelesaikan suatu masalah. Berapa besar masalah dapat diselesaikan ? Kolom (2) dari tabel menunjukkan secara kasar berapa besar masalah yang dapat diselesaikan masing-masing program dalam 1000 detik.

Tabel 1.4 *TABEL WAKTU*

Waktu Pelaksanaan T(n)	Besar Masalah untuk 1000 detik	10000 detik	Kenaikan Besar Masalah
100n	10	100	10
$5n^2$	14	45	3.2
$n^3 / 2$	12	27	2.3
2n	10	13	1.3

Andaikata sekarang kita mempunyai mesin lain yang dapat bekerja 10 kali lebih cepat tanpa tambahan biaya. Jadi untuk biaya yang sama, kita memiliki 10000 detik untuk menyelesaikan masalah yang tadinya tersedia waktu 1000 detik. Sekarang besarnya masalah yang dapat diselesaikan meningkat. Hal ini dapat dilihat pada kolom 3 dari tabel, sementara perbandingan peningkatannya terhadap mesin lama terlihat pada kolom 4.

Pada tabel, dapat kita lihat "kehebatan" program O(n). Dengan kenaikan kecepatan 10 kali, ternyata ia dapat menyelesaikan masalah 10 kali lebih besar. Sementara itu program $O(n^2)$ dan $0(n^3)$ hanya menghasilkan kenaikan 3.2 kali dan 2.3 kali. Pada program $O(2^n)$ ternyata hanya berhasil ditingkatkan besar masalah 1.3 kali untuk kenaikan kecepatan 10 kali. Berarti program $O(2^n)$ hanya mampu menyelesaikan masalah berukuran kecil bagaimanapun cepatnya komputer yang digunakan.

Sebagai kesimpulan dan catatan pembicaraan kita, maka beberapa hal perlu ditekankan dan ditambahkan di sini. Kalau program digunakan hanya beberapa kali, maka biaya pembuatan dan *debuging* program merupakan biaya utama. Jadi kita buat program yang rendah biaya pembuatannya. Biaya pelaksanaan/waktu komputer relatif tidak terpengaruh pada biaya total.

Kalau program diperuntukkan bagi input berukuran kecil kita harus memperhatikan faktor konstanta c dari fungsi waktu pelaksanan dibanding dengan laju kenaikan program.

Algoritma yang rumit dan berbelit-belit meskipun efektif sebaiknya tidak digunakan, karena menyulitkan dalam pemeliharaannya, kecuali oleh si pembuatnya. Ikutilah teknik yang prinsipil untuk algoritma yang efisien.

Terakhir sekali, perlu dicatat bahwa selain efisiensi, faktor ketelitian dan kestabilan merupakan faktor yang penting dalam algoritma untuk komputasi numerik.

L A T I H A N 1

1. Apa beda dari “data” dan “informasi,” serta berikan contoh-contohnya ?
2. Sebutkan tipe-tipe data yang Anda kenal ?
3. Sebutkan ukuran besaran data di komputer mulai yang terkecil (*bit*) hingga yang terbesar (*data bank*) ?
4. Sebutkan jenis-jenis struktur data yang Anda kenal ?
5. Apa beda tipe data *integer* dan *real,* berikan contoh-contohnya ?
6. Sebutkan macam-macam operasi yang dapat dilakukan terhadap data dengan tipe numerik ?
7. Sebutkan macam-macam operasi yang dapat dilakukan terhadap data dengan tipe karakter atau *string* ?
8. Pada operasi INDEX(‘My oldfather is the professor’, ‘the’) akan menghasilkan jawaban yang berupa tipe data numerik dengan nilai (a) 9, atau (b) 17 ?
9. Jika komputer yang Anda gunakan adalah komputer 16 bit, berapa batasan angka *integer* yang diterima komputer tersebut ?
10. Apa fungsi dari operator DIV dan MOD pada variabel *integer* ?
11. Apa yang terjadi ketika berita kesalahan *overflow error* muncul di monitor ?
12. Apa yang mungkin terjadi ketika berita kesalahan *devision by zerro error* muncul di monitor ?
13. Apa yang dimaksud dengan *floating point* dan berikan contohnya ?
14. Tuliskan cara pendeklarasian sebuah variabel (a) *integer*, (b) *real*, (c) *string* di dalam bahasa pemrograman (a) Pascal, (b) COBOL, dan (c) BASIC ?
15. Sekumpulan bit akan dijadikan sebuah *byte*. Sebutkan berbagai jenis kode yang mengartikan sekian bit (ada yang 4, 7, 8, dan sebagainya) menjadi sebuah *byte* yang Anda ketahui !
16. Apa guna melakukan penghitungan big-Oh ?, dan bagaimana cara melakukan perhitungannya ?
17. Ada 3 (tiga) kondisi waktu yang biasa dihitung dalam big-Oh, sebutkan ketiganya ?
18. Buatlah program dengan memanfaatkan operasi string untuk membuat teks berjalan (*running text*) di bagian bawah layar monitor Anda !