2 ARRAY (LARIK) DAN RECORD

Salah satu struktur data yang teramat penting adalah *array* atau larik. *Array* merupakan bagian dasar, yang disebut blok, guna keperluan pembentukan suatu struktur data lain yang lebih kompleks. Hampir setiap jenis struktur data kompleks dapat disajikan secara logik oleh *array*.

Kita dapat mendefinisikan *array* sebagai suatu himpunan hingga elemen, terurut dan homogen. Terurut, kita artikan bahwa elemen tersebut dapat diidentifikasi sebagai elemen pertama, elemen kedua, dan seterusnya sampai elemen ke-n. Sedangkan pengertian elemen yang homogen adalah bahwa setiap elemen dari sebuah *array* tertentu haruslah mempunyai tipe data yang sama.

Jadi suatu *array* dapat mempunyai elemen semuanya berupa *integer* atau dapat pula seluruhnya berupa untai aksara atau *string* Bahkan dapat pula terjadi bahwa suatu *array* mempunyai elemen berupa *array* pula.

Sebenarnya, pengertian array telah banyak kita kenal, dan kita pelajari dalam matematika. Di sana, *array* lebih terkenal sebagai matriks. Kadang-kadang ia disebut juga sebagai tabel. Juga pernah kita dengar tentang vektor. Vektor adalah bentuk yang paling sederhana dari *array*. Vektor merupakan *array* dimensi satu atau *one dimensional array*.

2.1 *ARRAY* DIMENSI SATU

Sebuah *array* dimensi satu, yang misalnya kita beri nama NILAI, dapat kita bayangkan berbentuk seperti Gambar 2.1.

Nilai(1)	Nilai(2)	Nilai(3)	- - -	Nilai(n)

Gambar 2.1. *Array berdimensi satu*

Subscript atau indeks dari elemen *array* menyatakan posisi, elemen pada urutan dalam *array* tersebut. Notasi yang digunakan bagi elemen *array*, biasanya adalah nama *array* dilengkapi dengan *subcript*.

Secara umum, suatu *array* dimensi satu A dengan tipe data T dan *subscript* bergerak dari L sampai dengan U, ditulis sebagai A(L:U) = (A(l)), I = L, L+1, L+2,..., U, dan setiap elemen A(l) bertipe data T.

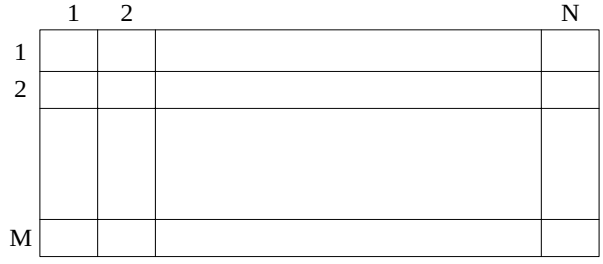
Sebagai contoh, kita dapat menuliskan data hasil pencatatan suhu suatu ruangan setiap satu jam selama periode 24 jam, dalam sebuah *array* dimensi satu.

Harga minimum dari *subscript* dari *array* disebut batas bawah atau *lower bound*, sedangkan harga maksimumnya disebut batas atas atau *upper bound*. Jadi pada *array* di atas, L merupakan batas bawah, dan U batas atas. Sedangkan untuk *array* "suhu" yang elemennya dapat kita tulis sebagai SUHU(I), batas bawahnya adalah 1 dan batas atasnya 24. SUHU(I) menyatakan suhu pada jam ke-1, dan I memenuhi 1 <= I <= 24, I merupakan *integer*.

Batas bawah dari *array*, pada beberapa aplikasi, tidak selalu diambil 1. Kadang-kadang diambil batas bawah nol, bahkan juga negatif. Banyaknya elemen sebuah *array* disebut rentang atau *range*. Jadi *array* A(L:U) mempunyai *range* sebesar U-L+1. Secara khusus bila L=l dan U=N, maka *range* dari *array* A(l:N) adalah N-I+1 = N.

2.2 *ARRAY* DIMENSI BANYAK

Sebuah *array* dimensi banyak atau *multi-dimensional array* didefinisikan sebagai sebuah *array* yang elemennya berupa *array* pula. Misal *array* B mempunyai M elemen berupa *array* pula, yang terdiri dari N elemen. Kalau hal tersebut kita gambarkan, akan terbentuk baris dan kolom seperti terlihat pada Gambar 2.2.



Gambar 2.2. *Array berdimensi dua*

Untuk itu diperlukan dua buah *subscript*. Yang pertama digunakan untuk menyatakan posisi baris, sedangkan yang kedua untuk posisi kolom. Secara umum *array* dimensi dua B, dengan elemen bertipe data T, *subscript* baris dari l sampai M, *subscript* kolom dari l sampai N, ditulis sebagai B(1:M, 1:N) = (B(I,J)), I = 1, 2, ...,M dan J = 1, 2,...,N dengan setiap elemen B(I,J) bertipe data T. *Array* B tersebut dikatakan berukuran atau berorder M x N. Di sini banyak elemen *array* adalah M*N.

Contoh dari *array* dimensi dua sangat banyak kita jumpai. Misalnya nilai ujian 500 mahasiswa Gunadarma tingkat 3, untuk 8 mata kuliah dapat kita sajikan sebagai *array* dimensi dua yang berorder 500 x 8. Elemen B(I,J) menyatakan nilai mahasiswa ke-I untuk mata kuliah ke-J.

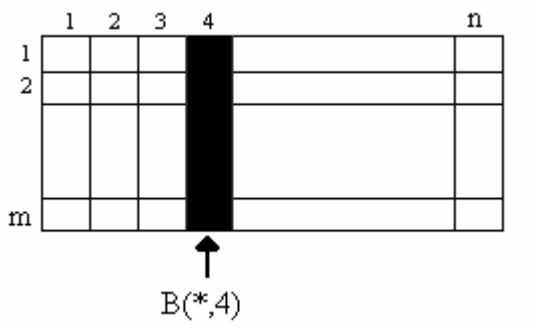
Seperti halnya pada *array* dimensi satu, pada *array* dimensi dua batas bawah untuk *subscript* I maupun J dapat diambil secara umum. Misalnya, batas bawah *subscript* baris adalah L_1 *subscript* kolom adalah L_2 sedangkan batas atas *subscript* baris adalah U_1 dan untuk kolom adalah U_2, maka *array* dimensi dua tersebut dapat dinotasikan sebagai :

$$B(L_1:U_1, L_2:U_2) = (B(I,J)),\ L_1 <= 1 <= U_1,\ L_2 <= J <= U_2$$

dengan setiap elemen B(I,J) bertipe data T. Banyaknya elemen pada setiap baris adalah $U_2 - L_2 + 1$ dan pada setiap kolom adalah $U_1 - L_1 + l$, sehingga banyaknya elemen pada *array* B semua ada = $(U_2-L_2 +1) * (U_1-L_1 +1)$.

Yang dimaksud dengan *cross-section* suatu *array* berdimensi dua adalah pengambilan salah satu *subscript*, misalnya *subscript* baris untuk tetap atau konstan, sementara *subscript* yang satunya lagi kita ubah-ubah sepanjang *range*nya. Notasi yang umum digunakan adalah notasi * (*asterisk*) bagi *subscript* yang berubah-ubah nilainya tersebut.

Contohnya, penulisan B(*,4) menyatakan semua elemen pada kolom ke-4, yakni (B(1,4),B(2,4), B(3,4) ...., B(M,4)), seperti terlihat pada Gambar 2.3



Gambar 2.3. *Cross section array*

Dengan mudah dapat dimengerti bahwa B(11,*) menunjukkan semua elemen pada baris ke-11.

Transpose dari suatu *array* dimensi dua adalah penulisan baris menjadi kolom (kolom menjadi baris) dari suatu *array*. Jadi *transpose* dari *array* berorder M x N adalah *array* berorder N x M. *Transpose* dari *array* B dinotasikan sebagai B^T. Berdasarkan definisi, maka jelas $B(I,J) = B^T(J,I)$. Contohnya $B(3,5) = B^T(5,3)$.

Pengertian di atas dapat kita perluas untuk *array* dimensi tiga, dimensi empat, sampai dimensi N. *Array* dimensi N kita tulis sebagai :

$$A(L_1:U_1, L_2:U_2, \ldots, L_N: U_N) = (A(I_1, I_2, \ldots, I_N))$$

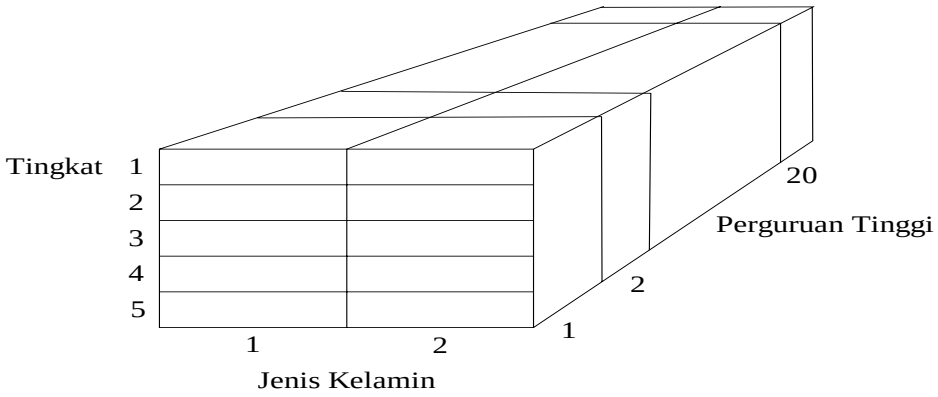
dengan $L_k <= I_k <= U_k$, untuk setiap k = 1, 2, …, N.

Banyaknya elemen dari *array* A tersebut adalah :

$$PI(Uk - Lk + 1) = (U_1-L_1+1) * (U_2 – L_2+1) \ldots * (U_N -L_N + 1)$$

Contoh *array* dimensi tiga adalah penyajian data mengenai banyaknya mahasiswa dari-20 perguruan tinggi di Jakarta, berdasarkan tingkat (tingkat 1, 2 sampai dengan 5), dan jenis kelamin (pria atau wanita). Misalnya *array* tersebut dinamakan MHS. Ambil sebagai *subscript* pertama, tingkat : I = 1, 2,...,5; *subscript* kedua, jenis kelamin (pria = 1, wanita = 2): J = 1,2, dan *subscript* ke-3, Perguruan Tinggi adalah K = 1,2,...,20. Jadi MHS(4,2,17) menyatakan jumlah mahasiswa tingkat 4, wanita, dari perguruan tinggi ke 17.

Array dimensi tiga dapat kita bayangkan seperti Gambar 2.4.



Gambar 2.4. *Array berdimensi tiga*

Pengertian *cross-section* pada *array* dimensi banyak, adalah sama seperti pada *array* dimensi dua. Misalnya MHS(4,*,17) menunjukkan jumlah mahasiswa tingkat 4 dari perguruan tinggi 17 (masing-masing untuk pria serta wanita). MHS(*,*,3) menun-jukkan jumlah mahasiswa untuk masing-masing tingkat, pria serta wanita, dari perguruan tinggi 3.

2.3 MENDEKLARASIKAN *ARRAY* DALAM BAHASA PEMROGRAMAN

Misalkan kita hendak mendeklarasikan *array* TEMP yang merupakan *array* dimensi satu dengan nilai *subscript* 1 sampai 24, dan masing-masing elemen bertipe data *integer* (nilainya antara 0 hingga 99 derajat).

Dalam Bahasa COBOL dapat ditulis :

```
01 TABEL-TEMP
     02 TEMP OCCURS 24 TIMES PIC 99.
```

Dalam bahasa Pascal :

```
var temp: array l..24)of integer
```

Dalam Bahasa BASIC, kita dapat mendefinisikan *array* TEMP tersebut dengan *statement* :

```
DIM  TEMP(24)
```

Tiga hal harus dikemukakan dalam mendeklarasikan suatu *array*, yakni :

1. nama *array*
2. *range* dari *subscript*
3. tipe data dari elemen *array*

Bahasa Pascal memperkenankan batas bawah *subscript* yang bukan =1, contohnya adalah :

```
var grafik : array [-100 ..100] of integer
```

Dalam COBOL *subscript* harus dimulai dari 1.

Untuk menyatakan elemen ke-I dari *array*, COBOL dan BASIC menggunakan kurung biasa, yakni TEMP(I), sedangkan Pascal menggunakan kurung siku, yakni temp[i].

Untuk mendeklarasikan sebuah *array* nilai dari 500 mahasiswa untuk 8 mata kuliah, dalam COBOL ditulis :

```
01 TABEL-NILAI
      02 MHS OCCURS 500 TIMES
                  03 NILAI OCCURS 8 TIMES
            PIC 99V9.
```

Dalam Pascal ditulis :

```
var nilai :  Array[1..500,1..8] of real
```

dan dalam BASIC dapat ditulis

```
      DIM  NILAI(500,8)
```

Dalam COBOL maksimum dimensi yang dapat diterima adalah 3 (*three dimensional*), contohnya :

```
01 MHS-TABEL
      02 TINGKAT OCCURS 5 TIMES
       03 SEX OCCURS 2 TIMES
            04 MHS OCCURS 20 TIMES  PIC 9(5).
```

dan dalam Pascal :

```
var mhs :  Array[1..5, 1..2, 1..20] of integer
```

Dalam bahasa pemrograman seperti FORTRAN dan COBOL, alokasi untuk *array* dalam *storage* memerlukan waktu dalam proses kompilasi, karenanya batas bawah dan batas atas harus dikemukakan ketika mendefinisikan *array*.

COBOL dan Pascal (juga bahasa lain yang memungkinkan pendeklarasian *array*) mempunyai fasilitas untuk melakukan manipulasi antarelemen *array*. Operasi yang sesuai dengan tipe data *array* tersebut dapat dikerjakan dengan mudah, contohnya dalam COBOL

```
COMPUTE  TOTAL_UPAH(I) = UPAH_PER_JAM(I) * JUMLAH-JAM(l)
```

Terlihat bahwa ketiga variabel di atas adalah *array*.

Beberapa bahasa pemrograman memperkenankan operasi *array*. Sebagai contoh, A adalah *array* (bertipe *real*) yang dideklarasi dalam PL/1, maka A=A+2 adalah operasi untuk menambah setiap elemen dari A dengan bilangan 2.

Juga dikenal operasi A = A * B. Operasi ini menghasilkan *array* A baru yang elemennya merupakan hasil kali elemen *array* A (lama) dengan elemen *array* B yang posisinya bersesuaian. Order *array* A dan B harus sama.

Perhatikan bahwa perkalian *array* ini bukan perkalian matriks yang telah kita kenal. Dalam PL/1, operasi dapat pula dilakukan terhadap *cross-section*. Sebagai contoh adalah operasi yang menyebabkan NILAI seluruh baris 20 menjadi nol, berikut ini :

```
NILAI(20,*)= 0
```

Operasi VEKTOR(*)= ARRAY1(I,*) *ARRAY(*,J) akan memperkalikan elemen baris ke-I dari ARRAY1 dengan elemen kolom ke-j dari ARRAY2. Operasi di atas mempunyai efek yang sama seperti *loop* dalam Bahasa BASIC :

```
FOR K = 1 TO N
VEKTOR(J) = ARRAY1(I,K)* ARRAY2(K,J)
NEXT K
```

2.4 PEMETAAN *ARRAY* DIMENSI SATU KE *STORAGE*

Seperti halnya struktur data yang lain, ada beberapa cara untuk menyajikan *array* di dalam memori. Skema penyajian dapat dievaluasi berdasarkan 4 karakteristik, yakni :

1. kesederhanaan dari akses elemen
2. mudah untuk ditelusuri
3. efisiensi dari utilitasi *storage*
4. mudah dikembangkan

Umumnya tidaklah mungkin untuk mengoptimalkan keempat faktor tersebut sekaligus. Pandang *array* satu dimensi NOPEG dengan batas bawah *subscript* 1, dan batas atas *subscript* = N. Salah satu cara untuk menyimpan *array* ini adalah sedemikian sehingga urutan fisik dari elemen sama dengan urutan logik dari elemen. *Storage* untuk elemen NOPEG(I+1) adalah berdampingan dengan *storage* untuk elemen NOPEG(I), untuk setiap I = 1, 2, 3,..., N-1. Untuk menghitung alamat (*address*) awal dari elemen NOPEG(I), diperlukan untuk mengetahui 2 hal yakni :

1. *address* awal dari ruang *storage* yang dialokasikan bagi *array* tersebut.
2. ukuran dari masing-masing elemen *array*.

Address awal dari *array*, kita nyatakan dengan B, disebut juga *base-location*. Misalkan bahwa masing-masing elemen dari *array* menduduki S *byte*. Maka, *address* awal dari elemen ke-I adalah :

$$B + (I-1) * S$$

Sekarang kita perluas persamaan di atas untuk mendapat *address* dari elemen ke-I dari a*rray* yang mempunyai batas bawah *subscript* tidak sama dengan 1. Perhatikan *array* Z(4:10), maka *address* awal dari Z(6) adalah :

$$B + (64) * S$$

Untuk *array* Z2 (-2:2) misalnya, *address* awal dari Z2(l) adalah :

$$B + (I -(-2)) * S$$

Maka secara umum, untuk *array* :

ARRAY(L:U),

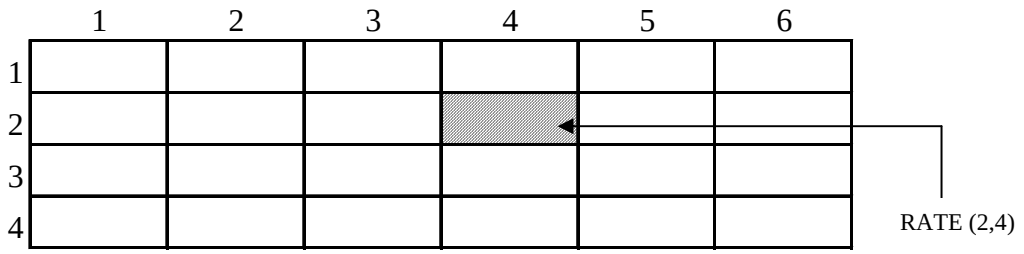
elemen ARRAY(I) mempunyai *address* awal

$$B + (U-L) * S$$

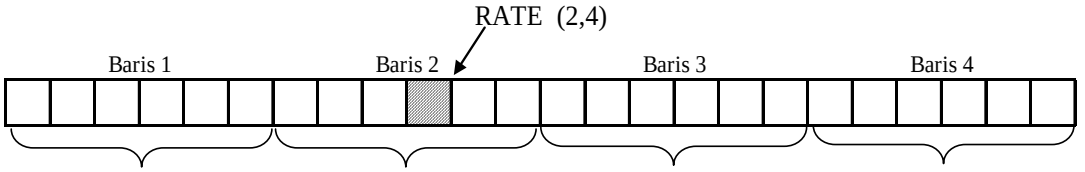
2.5 PEMETAAN KE *STORAGE* TERHADAP *ARRAY* DIMENSI BANYAK

Karena memori komputer adalah *linear*, maka *array* dimensi banyak harus di*linear*kan apabila akan dipetakan ke dalam *storage*. Salah satu alternatif untuk pe*linear*an tersebut adalah menyimpan pertama kali baris pertama dari *array*, kemudian baris ke-2, baris ke-3 dan seterusnya. Ini disebut *row major order*.

Sebagai contoh, *array* yang dideklarasikan sebagai RATE(1:4,1:6), yang secara logika tergambar sebagai pada Gambar 2.5 dan secara fisik tergambar pada Gambar 2.6.



Gambar 2.5. *Array RATE secara logik dalam row major order*



Gambar 2.6 *Array RATE secara fisik dalam row major order.*

Skema seperti di atas digunakan dalam COBOL, Pascal ataupun PL/1.

Misalkan B adalah *base-location* dari *array* RATE tersebut, dan masing-masing elemen dari *array* berukuran S. *Address* awal dari elemen RATE(I,J) adalah :

$$B + (I-1) * 6 * S + (J-1) * S$$

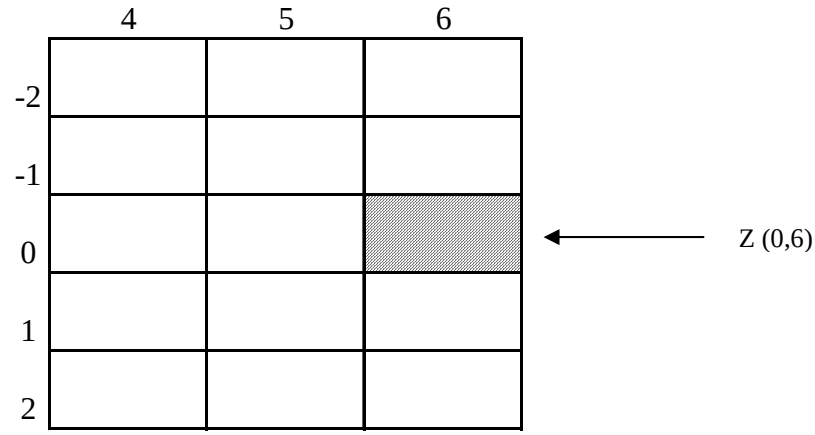
karena ada I-1 baris, masing-masing dengan panjang 6 * S, sebelum baris elemen RATE(I,J) terletak, dan terdapat J- 1 elemen, masing-masing dengan panjang S sebelum elemen RATE(I,J) pada baris ke-I. Jadi, pada contoh di atas RATE(2,4) mempunyai *address* awal :

$$B+ (2-1) * 6 * S + (4-1) * S = B + 9 * S$$

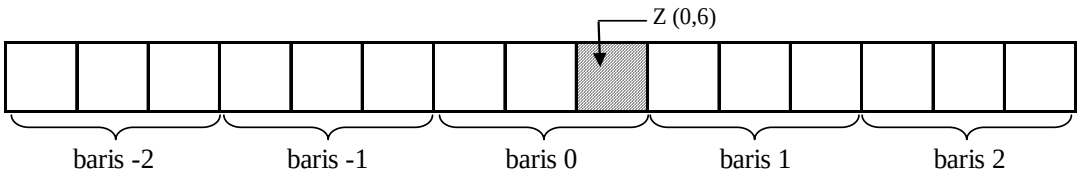
Secara umum elemen ARRAY(I,J) dari *array* yang didefinisikan sebagai ARRAY(L_1:U_1, L_2 : U_2) mempunyai *address* awal :

$$B + (I-L_1) * (U_2 -L_2+ 1) * S + (J-L_2) * S$$

Untuk lebih jelasnya, kita lihat *array* Z(-2:2, 4:6) yang dapat digambarkan pada Gambar 2.7 secara logik dan secara fisik dapat digambarkan pada Gambar 2.8.



Gambar 2.7. *Array Z (-2:2, 4:6) secara logik dalam row major order*

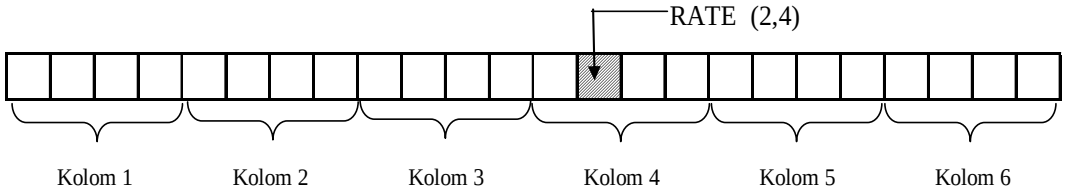


Gambar 2.8. *Array Z(-2:2,4:6) secara fisik dalam row major order*

Terdapat 2 baris (I-L_1, 0 – (-2)) sebelum baris nol, yang masing-masing panjangnya 3* S(U_2-L_2+1 = 6-4+1) dan terdapat 2 elemen (J-L_2 = 6-4) pada baris ke nol sebelum elemen Z (0,6). Jadi *address* awal dari Z(0,6) adalah :

$$B + 2 * 3 * S + 2 * S = B + 8 * S$$

Alternatif lain untuk me*linear*kan *array* dimensi dua adalah dengan menyimpan elemen dalam *column major order*, yakni pertama kali menyimpan kolom pertama, lalu kolom kedua, kolom ketiga dan seterusnya. *Array* RATE pada contoh di atas terlihat secara fisik dalam *column major order* seperti pada Gambar 2.9.



Gambar 2.9. *Array RATE secara fisik dalam column major order*

Skema seperti ini biasa digunakan dalam FORTRAN.

Dengan mudah dapat diterangkan bahwa pada *array* RATE di atas, elemen RATE(I,J) mempunyai *address* awal B + (J - 1) * 4 * S + (I - 1) * S, sehingga RATE(2,4) akan mempunyai *address* awal B + (4-1) * 4 * S + (2-1) * S = B + 13 * S. Jadi kita harus waspada andaikata kita mempunyai *array* yang ditulis dalam rutin FORTRAN, kemudian akan kita tulis dalam bahasa lain (COBOL, PL/1 atau Pascal). Secara umum, elemen ARRAY(I,J) dari *array* yang didefinisikan sebagai ARRAY(L_1:U_1,L_2 :U_2), menggunakan *address* awal :

$$B + (J-L_2) * (U_1-L_1 +1) * S + (I-L_1) * S$$

Misalkan *array* A berorder 50 x 225. Hendak dihitung jumlah / total elemennya. Kalau dipergunakan *column-major storage*, dapat kita buat, dalam COBOL.

```
COMPUTE TOTAL = 0.
PERFORM SUM-UP VARYING J
           FROM 1 BY 1 UNTIL J > 225
           AFTER 1 FROM 1 BY 1 UNTIL I > 50.
```

dalam hal ini

```
SUM-UP.
TOTAL = TOTAL + A(I,J).
```

Dalam Pascal dapat kita tulis :

```
Total := 0;
for j = 1 to 225 do
     for i = 1 to 50 do
      total := total + a[I,j];
```

Kalau dipergunakan *row-major storage*, kita dapat tulis dalam COBOL sebagai berikut :

```
COMPUTE TOTAL = 0.
PERFORM SUM-UP VARYING 1
     FROM 1 BY 1 UNTIL I > 50
     AFTER J FROM 1 BY 1 UNTIL J > 225
```

dalam hal ini

```
SUM-UP.
TOTAL  = TOTAL + A(I,J).
```

dan dalam Pascal dapat ditulis

```
total:=0;
for i := 1 to 50 do
     for j := 1 to 225 do
           total := total + a[i,j];
```

2.6 *TRINGULAR ARRAY* (*ARRAY* SEGITIGA)

Akan kita tinjau beberapa aspek pe*linear*an suatu *array* yang khusus, yakni *tringular array*. *Tringular array* dapat merupakan *upper tringular* (seluruh elemen di bawah diagonal utama = 0) ataupun *lower tringular* (seluruh elemen di atas diagonal utama = 0).

Dalam *array lower triangular* dengan N baris, jumlah maksimum elemen <> 0 pada baris ke-I adalah 1, karenanya total elemen <> 0, tidak lebih dari :

$$\sum_{I=1}^{N} I = N(N+1)/2$$

Gambar 2.10 menunjukkan *triangular array* berorder 6 x 6.

X	X	X	X	X	X		X	0	0	0	0	0
0	X	X	X	X	X		X	X	0	0	0	0
0	0	X	X	X	X		X	X	X	0	0	0
0	0	0	X	X	X		X	X	X	X	0	0
0	0	0	0	X	X		X	X	X	X	X	0
0	0	0	0	0	X		X	X	X	X	X	X
		(a)							*(b)*			

Gambar 2.10 *(a) Upper triangular array*
(b) Lower triangular array

Rumus ini berlaku pula untuk *array upper tringular* dengan N baris. Kalau N besar, alangkah baiknya kalau elemen nol tidak usah kita simpan dalam memori. Suatu pendekatan terhadap problema ini adalah dengan pe*linear*an *array*, dan dengan hanya menyimpan bagian *array* yang tidak nol.

Misalkan kita menyimpan *array upper tringular* T secara baris dalam *array* satu dimensi S, dengan batas *subscript* I sampai N(N+I)/2. Elemen T(1,1) disimpan sebagai S(1), elemen T(1,2) sebagai S(2) dan seterusnya, sehingga elemen T(1,N) disimpan sebagai S(N). Maka elemen T(2,2) disimpan sebagai S(N+1) (karena T(2,1) = 0). Terakhir sekali, elemen T(N,N) akan disimpan sebagai S(N(N+1)/2).

Kadang-kadang suatu program menggunakan lebih dari satu *array tringular*. Untuk itu kita dapat menyimpan 2 *array* sekaligus. Misalnya *array* A *upper triangular* berorder N x N dan *array* B *lower triangular* berorder (N-1) x (N-1). Mereka dapat kita simpan sebagai *array* C berorder N x N. Di sini C(l,J) = A(l,J) untuk I <= J dan C(I+1,J) = B(I,J) untuk I >= J.

Sekarang apabila *array* A *upper tringular* berorder N x N sedangkan *array* B *lower tringular*, juga berorder N x N, maka *array* C yang mengandung keduanya harus berorder N x (N+1). Di sini elemen A(I,J) disimpan sebagai C(I,J+1) untuk I <= J, dan B(I,J) disimpan sebagai C(I,J) untuk I >= J.

Perhatikan contoh berikut *array* A berorder (3 x 3) merupakan *upper tringular*.

```
1 2 3
0 4 5
0 0 6
```

Array B berorder (2 x 2) merupakan *lower tringular*,

```
7 0
8 9
```

maka mereka disimpan bersama dalam *array* C sebagai

```
1 2 3
7 4 5
8 9 6
```

Contoh berikut,

```
      1  2  3              7  0  0
A =   0  4  5        B =   8  9  0
      0  0  6             11 12 13
```

dapat disimpan sebagai *array* C berorder (3 x 4)

```
 7   1   2  3
 8   9   4  5
11  12  13  6
```

Misalkan sekarang ada 2 *array*, sama-sama *upper tringular*, yakni *array* A1 dan A2. Kita dapat menyimpan mereka bersama-sama dengan melakukan *transpose* terhadap salah satu *array* tersebut, misalnya A2 menjadi $A2^T$. $A2^T$ adalah *array lower tringular*. *Array* C berorder N x (N+1) akan menyimpan elemen A1(I,J) sebagai C(I,J+1) untuk I <= J, dan elemen A2(I,J) akan disimpan sebagai C(J,I) untuk I >= J.

Sebagai contoh adalah *array* :

```
      1  2  3              7  8  9
A1 =  0  4  5        A2 =  0 11 12
      0  0  6              0  0 13
```

```
maka A^2T =   7  0  0
              8 11  0
              9 12 13
```

dan mereka tersimpan sebagai :

```
      7   1   2  3
C =   8  11   4  5
      9  12  13  6
```

2.7 *SPARSE ARRAY* (*ARRAY* JARANG)

Suatu *array* yang sangat banyak elemen nolnya, dikenal sebagai *sparse array*, contohnya adalah *array* A pada Gambar 2-11 :

0	0	0	0	1	0	0	2	0	0
0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	4	0	0	0	0	0	0
0	0	0	0	0	0	0	2	0	0
0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0

Gambar 2-11. *Sparse array A*

Array hanya mempunyai 8 elemen yang bukan 0 dari 80 elemen yang ada. Kadang-kadang ada *array* berorder 1000 x 1000, yang hanya mengandung 1500 elemen bukan 0. Tentu saja akan sangat menguntungkan apabila kita cukup menyimpan elemen yang bukan 0 saja.

Ada dua alternatif yang dapat kita lakukan. Masing-masing elemen yang bukan 0 pada *array* berdimensi dua kita simpan sebagai tripel, dengan bentuk :

(*subscript* baris, *subscript* kolom, nilai elemen)

Tripel tersebut kita simpan sebagai vektor.

Sebagai contoh, *array* pada Gambar 2.11 dapat kita sajikan seperti pada Tabel 2.1

Tabel 2.1 *Array A disimpan sebagai tripel*

	Baris	Kolom	Nilai
V(I)	i	5	1
V(2)	1	8	2
V(3)	2	2	1
V(4)	3	1	1
V(5)	5	4	4
V(6)	6	8	2
V(7)	8	1	2
V(8)	8	2	1

Apabila *sparse array* tersebut adalah *array* berdimensi satu, maka masing-masing elemen <> 0 dinyatakan dalam pasangan. Secara umum untuk *array* berdimensi N, elemen <> 0 dinyatakan dengan tripel-N. Operasi terhadap vektor memperhatikan informasi tentang baris dan kolom.

Kekurangan dari penyajian ini adalah bila dilakukan *updating*. Elemen *array* yang tadinya <> 0, diubah menjadi 0 atau sebaliknya, yang tadinya 0 sekarang menjadi <> 0, menimbulkan kesulitan. Urutan dari vektor perlu diperbaiki. Misalnya elemen dengan *subscript* (1,8) di*update* nilainya menjadi bernilai 0, maka vektor V(3) hingga V(8) berubah urutannya sebagai vektor V(2) hingga V(7).

Juga misalnya elemen ber*subcript* (4,6) di*update* sehingga bernilai 7, maka vektor V(5) hingga V(8) berubah menjadi V(6) hingga V(9), sedangkan V(5) diisi oleh tripel (4,6,7). Penyajian yang lain dari *sparse array* adalah menggunakan daftar berkait atau *linked list*. Hal ini akan dibicarakan khusus dalam bab mengenai *Linked List.*

2.8 *RECORD*

Sebuah *record* merupakan koleksi satuan data yang heterogen, yakni terdiri dari berbagai *type*. Satuan data tersebut sering disebut sebagai *field* dari *record*. *Field* dipanggil dengan menggunakan namanya masing-masing. Suatu *field* dapat terdiri atas beberapa *subfield.*

Sebagai Contoh, data personalia dari seorang pegawai suatu perusahaan di Amerika Serikat, merupakan sebuah *record* yang dapat terdiri dari berbagai *field*, dan *subfield* seperti berikut ini :

```
1   NOMOR-JAMINAN-SOSIAL
2     NAMA, yang terdiri atas :
      NAMA-BELAKANG
      NAMA-DEPAN
      NAMA-TENGAH
3  ALAMAT, terdiri atas :
      JALAN
      NOMOR RUMAH
      NAMA-JALAN
      KOTA
      NEGARA-BAGIAN
      KODE-POS
4  MENIKAH
```

dan sebagainya lagi.

Pada *record* tersebut di atas, satuan data seperti NAMA BELAKANG ataupun KOTA merupakan tipe data *string*, sedangkan data lain seperti GAJI POKOK, TUNJANGAN JABATAN dan berbagai data yang akan diolah secara matematis akan disimpan dengan tipe data numerik, bisa *integer* maupun *real*. Data MENIKAH bisa digunakan tipe data *boolean* atau logikal.

Seperti telah kita paparkan terdahulu, *array* berbeda dengan *record*, yakni *array* bersifat homogen (terdiri dari tipe data yang sama), dan komponen *array* tidak memiliki nama sendiri, dan hanya diberi identifikasi oleh posisi mereka di dalam *array*. Penggunaan keduanya di dalam program juga berbeda, jika penggunaan *array* pada umumnya akan disimpan di memori utama komputer (bersifat sementara), sedangkan *record* biasanya digunakan dalam *filing* yang akan disimpan di memori sekunder komputer, seperti *hard disk*, disket, dan lainnya.

Sebuah *record* memberi informasi tentang berbagai kondisi dari obyek pada permasalahan yang nyata sehari-hari. Setiap *field* memberi uraian tentang satu atribut dari obyeknya. Sebuah *record* biasanya diberi identifikasi oleh *key*-nya. *Key* atau kunci adalah salah satu atau lebih *field* yang dipilih untuk tujuan penyampaian informasi yang terjadi di dalam *record* yang bersangkutan.

Koleksi dari *record* yang sama struktur *field*nya disebut suatu *file* atau berkas. Jadi, koleksi dari *record* semua pegawai perusahaan membentuk sebuah *file* personalia. Pada umumnya *record* disimpan membentuk *file*, dalam urutan sesuai dengan nilai dari *key* masing-masing. Di dalam suatu *file* PERSONALIA, *field* NOMOR JAMINAN SOSIAL dari seorang pegawai dapat digunakan sebagai *key*. Di dalam bahasa pemrograman tingkat tinggi, *record* dapat dinyatakan sebagai struktur data (COBOL dan PL/1) dapat diadakan spesifikasi tentang nama *record*, *field* dan *subfield* yang bersangkutan.

Record tersebut juga diberi nomor seperti diperlihatkan di dalam contoh di bawah ini. Deklarasi berikut ini dapat digunakan untuk menuliskan *record* dari *file* PERSONALIA di atas.

```
01 PEGAWAI
   02 NOMOR-JAMINAN-SOSIAL
   02 NAMA
      03 NAMA-BELAKANG
      03 NAMA-DEPAN
      03 NAMA-TENGAH
   02 ALAMAT
      03 JALAN
           04  NOMOR RUMAH
           04 NAMA-JALAN
      03 KOTA
      03 NEGARA-BAGIAN
      03 KODE-POS
   02 MENIKAH
```

(yang harus dilengkapi dengan *Picture* masing-masing *field* dan *subfield*)

Record tersebut dinyatakan di dalam memori sebagai berikut :

NOMOR JAM-SOS	NAMA BLK.	NAMA DEPAN	NAMA TENG.	NOMOR RUMAH	NAMA JALAN	KOTA	NEG. BAGIAN	KODE POS	MENI-KAH

Secara fisik, *field record* tersebut biasanya disimpan berurutan di dalam lokasi *storage*, bahkan sering disatukan. *Record* biasanya disimpan sebagai *file* di dalam *storage* pembantu, dan jika perlu, sebagian disimpan di dalam memori utama. *File* merupakan organisasi data utama di dalam proses pengolahan informasi.

Sebagai gambaran sederhana, pandang sebuah tabel dengan sejumlah baris dan kolom. Tabel tersebut dapat disebut sebagai sebuah *file*, sedangkan setiap baris dari tabel tersebut disebut dengan *record,* dan setiap kolom dari tabel disebut dengan *field.*

Tabel 2.2 *Contoh sebuah file TEMAN*

	NPM	NAMA	ALM_TEMAN	NO_TELP
record →	10102456	Cheriwaty	Jl. Puspa No. 30, Keb. Baru	7750658
record →	10102587	Tia Siti Joy	Jl. Veteran No. 38, Tebet	8652541
record →	10102965	Yekti Sartanto	Jl. Mampang No. 29, Jaksel	78956215
	field	*field*	*field*	*field*

Pembahasan mendalam tentang *file* akan dibahas di mata kuliah-mata kuliah yang memiliki sub-bahasan mengenai pengorganisasian dan pengaksesan *file*, perancangan sistem, perancangan *data base,* dan sejenisnya.

L A T I H A N 2

1. Apa yang dimaksud dengan istilah (a) *lower bound*, *upper bound*, dan (c) *cross-section* dalam suatu *array* ?

2. Jelaskan cara mendeklarasikan variabel *array* di bahasa pemrograman (a) Pascal (b) BASIC, dan (c) COBOL ?

3. Beda *array* dengan *record* adalah, isi dari variabel *array* harus bersifat (a) homogen atau (b) heterogen. Jelaskan apa maksud dari homogen dan heterogen tersebut.

4. Kita memiliki matriks M berukuran 200 X 300 yang akan kita simpan ke dalam memori komputer. Jika elemen awal matriks M(1,1) disimpan di alamat 1000 dan setiap elemen membutuhkan 8 *byte*, di alamat mana awal disimpannya elemen M(191, 280) ? Gunakan perhitungan secara (a) *row major order* dan (b) *colomn major order*.

5. Apa yang dimaksud dengan istilah (a) *tringular array*, (b) *upper tringular*, (c) *lower triangular*, dan (d) *sparse* dalam *array* ?

6. Jelaskan cara untuk menghemat ruang memori untuk menyimpan matriks yang bersifat *sparse array*.

7. Berapa jumlah minimal *record* di dalam sebuah *file* ?

8. Pada umumnya, *file* data digunakan untuk menyimpan data apa ?

9. Buatlah program dengan memanfaatkan *array* untuk menghitung 100 buah padanan suhu dari Celcius ke Fahrenheit dan Reamur.