Larik

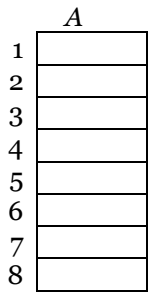
Apakah Larik Itu?

Larik adalah struktur data yang menyimpan sekumpulan elemen yang bertipe sama, setiap elemen diakses langsung melalui indeksnya. Indeks larik haruslah tipe data yang menyatakan keterurutan, misalnya *integer* atau karakter.

Sebuah larik yang bernama *A* dengan delapan buah elemen dapat dibayangkan secara lojik sebagai sekumpulan kotak yang terurut (baik tersusun secara vertikal atau horizontal) seperti yang diperlihatkan pada Gambar 12.1. Tiap kotak pada larik tersebut diberi indeks 1, 2, 3, ..., 8. Setiap elemen larik ditulis dengan notasi:

$A[1], A[2], A[3], A[4], A[5], A[6], A[7], A[8]$

Angka di dalam tanda kurung siku menyatakan indeks larik (notasi di atas sama saja dengan penulisan peubah ber-*subscript* $A_1, A_2, A_3, \ldots, A_8$).



Setiap elemen larik menyimpan sebuah nilai. Karena seluruh elemen larik bertipe sama, maka nilai yang disimpan oleh setiap elemen juga harus bertipe sama.

berisi nilai.

Mendeklarasikan Larik

Larik adalah struktur data yang *statis*, artinya jumlah elemen larik harus sudah diketahui sebelum program dieksekusi. Jumlah elemen larik tidak dapat diubah, ditambah, atau dikurangi selama pelaksanaan program. Mendeklarasikan larik di dalam bagian deklarasi berarti:

1. mendefinisikan banyaknya elemen larik (ukuran larik), dan

2. mendefinisikan tipe elemen larik.

Mendefinisikan banyaknya elemen larik (atau ukuran larik) berarti memesan sejumlah tempat di memori. Komputer mengalokasikan sejumlah lokasi memori sebanyak elemen larik yang bersangkutan. Mendefinisikan tipe elemen larik berarti menetapkan tipe nilai yang dapat disimpan oleh larik. Tipe elemen larik dapat berupa tipe sederhana (*integer*, *real*, *char*, *boolean*, *string*), tipe bentukan (tipe terstruktur seperti *record*), atau bahkan bertipe larik yang lain.

Berikut contoh-contoh mendeklarasikan larik di dalam bagian deklarasi:

a. Sebagai Peubah

Misalkan:

- `A` adalah larik yang berukuran 50 buah elemen yang bertipe *integer*. Indeks larik dimulai dari 1.
- `NamaMhs` adalah larik yang berukuran 10 buah elemen yang bertipe *string*. Indeks larik dimulai dari 1.
- `NilUjian` adalah peubah larik yang berukuran 75 buah elemen yang bertipe *real*. Indeks larik dimulai dari 0.

Cara mendefinisikan ketiga buah peubah larik tersebut di dalam bagian deklarasi adalah:

```
DEKLARASI
  A        : array[1..100] of integer
  NamaMhs  : array[1..12] of string
  NilUjian : array[0..74] of real
```

```
DEKLARASI
  const Nmaks = 100   { ukuran maksimum elemen larik }
  type LarikInt : array[1..Nmaks] of integer
  A : LarikInt { A adalah sebuah peubah larik integer dengan 100 elemen }
```

Cara Mengacu Elemen Larik

Elemen larik diacu melalui indeksnya. Nilai indeks harus terdefinisi. Dengan mengacu pada larik yang sudah dideklarasikan sebelum ini, berikut diberikan beberapa contoh cara mengacu elemen larik adalah:

```
A[4]                  { megacu elemen keempat dari larik A }
NamaMhs[2]            { mengacu elemen kedua dari larik NamaMhs }
A[i]                  { mengacu elemen ke-i dari larik A, asalkan
                        nilai i sudah terdefinisi }
NamaMhs[i+1]          { asalkan nilai i sudah terdefinisi }
```

Contoh-contoh memanipulasi atau menggunakan elemen larik:

```
A[4] ← 10              { mengisi elemen keempat dari larik A dengan
                         nilai 10 }

NamaMhs[i] ← 'Achmad' { mengisi elemen ke-i dari larik NamaMhs
                        dengan string 'Achmad' }
read(A[i])             { membaca elemen ke-i dari larik A }

if A[i] < 10 then
  A[i]←A[i] + 10
else
   ... { pernyataan lainnya }
```

Pemrosesan Larik

Elemen larik tersusun di memori secara beruntun (sekuensial). Karena itu, elemennya diproses secara beruntun melalui indeksnya yang terurut. Memproses larik artinya mengunjungi (traversal) setiap elemen larik dan memanipulasi dinilai di dalamnya. Kunjungan dimulai dari elemen pertama larik, berturut-turut pada elemen berikutnya, sampai elemen terakhir dicapai, yaitu elemen dengan indeks terbesar.

Skema umum algoritma memproses larik diperlihatkan pada Algoritma 12.5.

```
PROGRAM PemrosesanLarik
{ Skema pemrosesan larik secara beruntun }

DEKLARASI
   const Nmaks = 100  { ukuran maksimum larik }
   type LarikInt  : array[1..Nmaks] of integer

   A : LarikInt
   i : integer     { indeks larik }

ALGORITMA:
  Inisialisasi
  i ← 1       { mulai dari elemen pertama }
  while i ≤ Nmaks do
     pemrosesan terhadap A[i]
     i ← i + 1  { tinjau elemen berikutnya }
  endwhile
  { i > Nmaks }
  Terminasi
```

Pemrosesan terhadap $A[i]$ adalah aksi spesifik bergantung pada persoalan yang akan dipecahkan, misalnya pengisian nilai, pembacaan, penulisan, komputasi, atau manipulasi lainnya.

Oleh karena jumlah elemen larik sudah diketahui di awal proses, maka jumlah pengulangan juga dapat ditentukan. Oleh karena itu, kita lebih menyukai struktur *FOR* digunakan untuk untuk memproses larik, seperti ditunjukkan pada Algoritma 12.6.

```
PROGRAM PemrosesanLarik
{ Skema pemrosesan larik secara beruntun }

DEKLARASI
   const Nmaks = 100  { ukuran maksimum larik }
   type LarikInt  : array[1..Nmaks] of integer

   A : LarikInt
   i : integer     { indeks larik }

ALGORITMA:
  for i ← 1 to Nmaks do
     pemrosesan terhadap A[i]
  endfor
```

Ukuran Efektif Larik

Meskipun kita mendefinisikan jumlah elemen larik, seringkali kita tidak menggunakan semuanya. Bila larik *A* didefinisikan 100 elemen, mungkin tidak seratus elemen yang dipakai, mungkin hanya 15, 40, atau 70. Banyaknya elemen larik yang dipakai kita sebut *ukuran efektif larik*. Ukuran efektif itu kita catat di dalam peubah tertentu, misalnya *n*.

Di bawah ini diberikan beberapa algoritma pemrosesan larik yang disajikan dalam bentuk prosedur. Larik yang digunakan adalah larik yang bernama *A* dan bertipe *integer*. Kita asumsikan ukuran maksimum elemen larik adalah 100, dan ukuran efektif larik yang digunakan kita simpan di dalam peubah *n*. Selain itu, skema yang digunakan menggunakan struktur *FOR*, dan pada masalah tertentu menggunakan struktur *WHILE*.

```
DEKLARASI
  const Nmaks = 100   { ukuran maksimum larik }
  type LarikInt : array[1..Nmaks] of integer
  A             : LarikInt
  n             : integer      { mencatat ukuran larik yang digunakan }
```

Menginisialisasi Larik

Menginisialisasi larik adalah memberikan nilai awal untuk seluruh elemen larik atau mungkin sebagian saja. Inisialisasi larik mungkin diperlukan, misalnya "mengosongkan" elemen larik sebelum dipakai untuk proses tertentu, tetapi hal ini bukan keharusan, bergantung pada permasalahan yang akan dipecahkan. "Mengosongkan" larik bertipe numerik dapat dilakukan dengan

mengisi seluruh elemen dengan nol (atau nilai lainnya, bergantung kebutuhan), sedangkan pada larik karakter, "mengosongkan" larik berarti mengisi elemen larik dengan spasi atau karakter kosong (*null*).

a. Menginisialisasi elemen-elemen larik dengan nilai 0

```
procedure InisDengan0(output A : LarikInt, input n : integer)
{ Menginisialisasi setiap elemen larik A[1..n] dengan nol. }
{ K.Awal: n adalah jumlah elemen efektif larik, nilainya terdefinisi.  }
{ K.Akhir: seluruh elemen larik A bernilai nol. }

DEKLARASI
   i : integer   { pencatat indeks larik }

ALGORITMA:
   for i ← 1 to n do
      A[i] ← 0
   endfor
```

Catatan:
untuk mempersingkat penulisan, maka larik A yang berukuran n elemen dan diacu dengan indeks dari 1 sampai n kita tulis sebagai A[1..n].

Contoh program yang memanggil `InisDengan0`:

```
PROGRAM PemrosesanLarik
{ Program untuk mengisi elemen larik dengan nilai 0 }

DEKLARASI
   const Nmaks = 100  { ukuran maksimum larik }
   type LarikInt  : array[1..Nmaks] of integer

   A  : LarikInt
   i  : integer   { indeks larik }
   n  : integer   { ukuran efektif larik }

   procedure InisDengan0(output A : LarikInt, input n : integer)
   { Menginisialisasi setiap elemen larik A[1..n] dengan nol. }

ALGORITMA:
   read(n)          { tentukan jumlah elemen larik yang akan digunakan,
                      dengan syarat 1 ≤ n ≤ Nmaks }
   InisDengan0(A, n)

   { cetak hasil inisialisasi }
   for i ← 1 to n do
     write(A[i])
   endfor
```

b. Menginisialisasi setiap elemen larik ke-*i* dengan nilai *i*.

Misalkan kita ingin setiap elemen larik yang ke-*i* diinisialisasi dengan nilai *i*. Jadi, *A*[1] = 1, *A*[2] = 2, ..., *A*[*n*] = *n*. Algoritma inisialisasinya adalah seperti berikut ini:

```
procedure InisDengan(output A : LarikInt, input n : integer)
{ Menginisialisasi setiap elemen A[i] dengan nilai i = 1, 2, ..., n. }
{ K.Awal: n adalah jumlah elemen efektif larik, nilainya terdefinisi. }
{ K.Akhir: A[1] = 1, A[2] = 2, ..., A[n] = n }

DEKLARASI
   i : integer { pencatat indeks larik }

ALGORITMA:
   for i ← 1 to n do
     A[i] ← i
   Endfor
```

Mengisi Elemen Larik dengan Pembacaan

Selain dengan pengisian nilai secara langsung (seperti pada inisialisasi), elemen larik diisi dengan cara pembacaan (misalnya dengan mengetikkan nilainya dari papan ketik). Dalam hal ini, elemen larik dibaca satu per satu mulai untuk elemen pertama sampai elemen ke-*n*.

Algoritma pembacaan elemen larik adalah seperti di bawah ini (kita buat dalam 3 versi sesuai spesifikasinya):

Versi 1: jika jumlah elemen efektif ditentukan di awal

```
procedure BacaLarik1(output A : LarikInt, input n : integer)
{ Mengisi elemen-elemen larik A[1..n] dengan pembacaan. }
{ K.Awal: n adalah jumlah elemen efektif larik, nilainya terdefinisi. }
{ K.Akhir: setelah pembacaan, seluruh elemen larik A berisi nilai-
  nilai yang dibaca dari piranti masukan. }

DEKLARASI
   i : integer   { pencatat indeks larik }

ALGORITMA:
   for i ← 1 to n do
      read(A[i])
   endfor
```

Versi 2: jika jumlah elemen efektif baru diketahui di akhir pembacaan

Adakalanya jumlah elemen efektif tidak ditentukan di awal, tetapi baru diketahui pada akhir pembacaan. Dalam hal ini, jumlah elemen efektif, *n*, merupakan parameter bertipe keluaran. Prosedur BacaLarik2 di bawah ini membaca elemen-elemen larik, setiap kali selesai pembacaan satu buah

elemen, pengguna program dikonfirmasi apakah masih ada lagi elemen larik yang akan dimasukkan.

```
write('Lagi?(y/t)')
read(jawab)
```

Jika jawabannya 'y' maka pembacaan dilanjutkan, jika 't' maka proses pembacaan dihentikan. Jumlah elemen yang telah dibaca dicatat di dalam peubah *n*.

```
procedure BacaLarik2(output A : LarikInt, output n : integer )
{ Mengisi elemen-elemen larik A[1..n] dengan cara pembacaan. }
{ K.Awal: sembarang. }
{ K.Akhir: sebanyak n buah elemen larik A berisi nilai-nilai yang
  dibaca; n berisi jumlah elemen larik yang diisi. }

DEKLARASI
   jawab : char

ALGORITMA:
   N ← 0
   repeat
     n ← n + 1
     read(A[n])
     write('Lagi?(y/t)')
     read(jawab)
   until jawab = 't'
```

***Versi 3*: jika jumlah elemen efektif baru diketahui di akhir pembacaan (variasi dari versi 3)**

Pada versi 3 ini, proses pembacaan dianggap selesai jika nilai yang dibaca adalah suatu tanda, misalnya 9999.

```
procedure BacaLarik3(output A : LarikInt, output n : integer )
{ Mengisi elemen-elemen larik A[1..n] dengan cara pembacaan. Akhir
  pembacaan ditandai jika nilai yang dibaca adalah 9999 }
{ K.Awal: sembarang. }
{ K.Akhir: sebanyak n buah elemen larik A berisi nilai-nilai yang
  dibaca; n berisi jumlah elemen larik yang diisi. }

DEKLARASI
   x   : integer   { menyimpan sementara nilai yang dibaca }

ALGORITMA:
   n ← 0
   read(x)
   while x ≠ 9999 do
     n ← n + 1
     A[n] ← x
     read(x)
   endwhile
   { x = 9999 }
```

Mencetak Elemen-elemen Larik

Isi elemen larik dicetak ke piranti dengan pernyataan `write`. Dalam hal ini, elemen larik dicetak satu per satu mulai untuk elemen pertama sampai elemen ke-*n*.

Algoritma pencetakan elemen-elemen larik dinyatakan oleh prosedur di bawah ini.

```
procedure CetakLarik(input A : LarikInt, input n : integer )
{ Mencetak elemen-elemen larik A[1..n]. }
{ K.Awal: n sudah berisi jumlah elemen larik yang dipakai. Elemen-
elemen larik A sudah terdefinisi.}
{ K.Akhir: elemen-elemen larik A tercetak. }

DEKLARASI
   i : integer  { pencatat indeks larik }

ALGORITMA:
   for i ← 1 to n do
      write(A[i])
   endfor
```

Kapan Menggunakan Larik?

Pertanyaan yang sering diajkan oleh orang yang baru belajar pemrograman adalah: kapan menggunakan larik? Larik digunakan bila kita mempunyai sejumlah data yang bertipe sama, dan kita perlu perlu menyimpan sementara data tersebut, untuk selanjutnya data tersebut kita proses.

Latihan-latihan:

1. Menghitung nilai rata-rata sejumlah bilangan
2. Mencari nilai terbesar atau terkecil.
3. Menyalin isi larik A ke larik B
4. Menguji kesamaan dua buah larik

Larik Bertipe Terstruktur

Contoh-contoh algoritma yang dibahas sebelum ini menggunakan larik dengan elemen-elemen bertipe sederhana. Elemen larik juga dapat bertipe terstruktur.

Sebagai contoh, misalkan kita akan mengolah data 100 orang mahasiswa. Data setiap mahasiswa terdiri dari NIM (Nomor Induk Mahasiswa), nama mahasiswa, dan IPK (indeks prestasi kumulatif yang nilainya berkisar antara 0 sampai 4). Struktur lojik larik *Mhs* ditunjukkan pada Gambar 12.4.

Struktur lojik larik *Mhs* ditunjukkan pada Gambar 12.4.

	NIM	*NamaMhs*	*IPK*
1	29801	Heru Satrio	3.04
2	29804	Amirullah Satya	2.75
3			
4			
5			
6			
7			
.			
.			
100	29887	Yanti Siregar	2.19

Struktur larik yang menyimpan keseluruhan data tersebut dideklarasikan di bawah ini. Nama larik adalah *Mhs*.

```
DEKLARASI
  const Nmaks = 100
  type Mahasiswa : record <NIM  : integer,  { Nomor Induk Mahasiswa }
                           NamaMhs : string,   { nama mahasiswa }
                           IPK  : real      { 0.00 sampai 4.00 }
                          >

  type TabMhs : array[1..Nmaks] of Mahasiswa
  Mhs   : TabMhs
```

Cara mengacu elemen Mhs:

```
Mhs[2]          { elemen kedua dari larik Mhs }
Mhs[2].NIM      { mengacu field NIM dari elemen kedua larik }
Mhs[2].IPK      { mengacu field IPK dari elemen kedua larik }
```

Karena *record* merupakan struktur, maka kita tidak dapat melakukan pencetakan elemen larik *Mhs* seperti di bawah ini:

```
write(Mhs[i])
```

tetapi haruslah seperti berikut ini:

```
write(Mhs[i].NIM, Mhs[i].NamaMhs, Mhs[i].IPK)
```

Namun, cara mengisi nilai elemen ke elemen lainnya dengan penulisan pernyataan seperti di bawah ini adalah benar:

```
        Mhs[i] ← Mhs[i+1]
```

yang sama saja dengan pengisian per *field* sebagai berikut:

```
Mhs[i].NIM ← Mhs[i+1].NIM
Mhs[i].NamaMhs ← Mhs[i+1].NamaMhs
```

```
Mhs[i].KodeMK ← Mhs[i+1].IPK
```

Algoritma untuk mengisi larik `Mhs` adalah seperti di bawah ini:

```
procedure BacaDataMahasiswa(input n : integer, output Mhs : TabMhs)
{ Membaca data mahasiswa (NIM, nama, IPK).
{ K.Awal  : n berisi jumlah data mahasiswa }
{ K.Akhir : Mhs berisi data hasil pembacaan }

DEKLARASI
   i : integer     { pencatat indeks larik }

ALGORITMA:
   for i←1 to N do
      read(Mhs[i].NIM)
      read(Mhs[i].NamaMhs)
      read(Mhs[i].IPK)
   endfor
```

Selain bertipe terstruktur, elemen larik juga dapat bertipe larik lain. Contoh berikut menyajikan struktur tipe bentukan yang cukup kompleks. Misalkan kita ingin mengolah data nilai semua mata kuliah yang diambil setiap mahasiswa di semester genap. Asumsikan setiap mahasiswa mengambil 4 buah mata kuliah yang berbeda. Setiap elemen larik berisi data sebagai berikut:

1. *NIM* (Nomor Induk mahasiswa)
2. *NamaMhs* (nama mahasiswa)
3. Mata kuliah (*MK*) yang diambil mahasiswa tersebut (4 buah), berupa larik:
   a. Kode mata kuliah ke-1
      Nama mata kuliah ke-1
      Nilai mata kuliah ke-1

   b. Kode mata kuliah ke-2
      Nama mata kuliah ke-2
      Nilai mata kuliah ke-2
   c. Kode mata kuliah ke-3
      Nama mata kuliah ke-3
      Nilai mata kuliah ke-3
   d. Kode mata kuliah ke-4
      Nama mata kuliah ke-4
      Nilai mata kuliah ke-4

Struktur larik yang menyimpan keseluruhan data tersebut dideklarasikan di bawah ini. Nama larik adalah *Mhs2*.

```
DEKLARASI
  const Nmaks = 100
  type MataKuliah : record <KodeMK : string, { kode mata kuliah }
                            Nilai  : char  { indeks nilai: A/B/C/D/E)}
                           >
```

```
type Mahasiswa  : record <NIM  : integer,     { Nomor Induk Mahasiswa }
                          NamaMhs : string,   { nama mahasiswa }
                          MK    : array[1..4] of MataKuliah
                         >
type TabMhs : array[1..Nmaks] of Mahasiswa
Mhs2   : TabMhs
```

Cara mengacu elemen `Mhs2`:

```
Mhs2[2]                { elemen kedua dari larik Mhs2 }
Mhs2[2].NIM            { mengacu field NIM dari elemen kedua larik }
Mhs2[2].MK[3].KodeMK   { mengacu field KodeMK ketiga dari elemen kedua
                         larik Mhs2 }
Mhs2[2].MK[3].Nilai    { mengacu field Nilai ketiga dari elemen kedua
                         larik Mhs2 }
```

Contoh algoritma untuk mengisi larik `Mhs2`:

```
procedure BacaDataMahasiswa(input n : integer, output Mhs2 : TabMhs)

{ Membaca data mahasiswa (NIM, nama, daftar mata kuliah, nilai tiap
mata kuliah }
{ K.Awal  : n berisi jumlah data mahasiswa }
{ K.Akhir : Mhs2 berisi data hasil pembacaan }

DEKLARASI
  i,j : integer    { pencatat indeks larik }

ALGORITMA:
  for i ← 1 to n do
      read(Mhs2[i].NIM)
      read(Mhs2[i].NamaMhs)
      for j ←1 to 4 do
         read(Mhs2[i].MK[j].KodeMK)
         read(Mhs2[i].MK[j].Nilai)
      endfor
   endfor
```

String sebagai Larik Karakter

Tipe *string* sudah kita pelajari ketika membicarakan tipe data yang diprogram. *String* pada hakikatnya adalah larik karakter dengan panjang dinamis, artinya ukuran larik baru ditentukan pada saat program *running*. Karena *string* adalah larik, maka elemen-elemen *string* yang berupa karakter dapat diakses melalui indeks. Sebagai contoh, jika *s* adalah peubah bertipe *string* dan *s* berisi konstanta *string* berikut

```
'ini string'
```

maka dapat kita katakan bahwa s panjangnya 10 karakter dan elemen-elemennya adalah:

```
s[1] = 'i'
s[2] = 'n'
s[3] = 'i'
s[4] = ' '
s[5] = 's'
s[6] = 't'
s[7] = 'r'
s[8] = 'i'
s[9] = 'n'
s[10] = 'g'
```

Representasi *string* bergantung pada bahasa pemrograman yang digunakan.

Soal Latihan Array

1. Diberikan larik *integer A* yang berukuran *n* elemen. Larik *A* sudah terdefinisi elemen-elemennya. Tuliskan prosedur untuk mencari nilai *X* di dalam larik, dengan parameter keluaran adalah indeks dari kemunculan *terakhir* elemen *X*. Jika *X* tidak terdapat di dalam larik, prosedur mengeluarkan indeks 0.

2. Diberikan larik *integer A* yang berukuran *n* elemen. Larik *A* sudah terdefinisi elemen-elemennya. Tuliskan prosedur yang keluarannya adalah elemen terbesar di dalam larik *A*.

3. Diberikan larik *integer A* dan *integer* larik *B* yang masing-masing berukuran *N* elemen. Larik *A* dan *B* sudah terdefinisi elemen-elemennya. Tuliskan prosedur untuk mempertukarkan elemen larik *A* dan elemen *B* pada indeks yang bersesuaian, sedemikian sehingga larik *A* berisi elemen-elemen larik *B* semula dan larik *B* berisi elemen-elemen larik *A* semula.

4. Diberikan larik karakter *A* yang berukuran *n* elemen. Larik *A* sudah terdefinisi elemen-elemennya. Tuliskan prosedur yang membalikkan elemen-elemen larik *A* sedemikian sehingga elemen terakhir pada larik semula menjadi elemen pertama pada larik akhir.
   Contoh:

sebelum pembalikan:	*m*	*a*	*r*	*a*	*h*

setelah pembalikan:	*h*	*a*	*r*	*a*	*m*