Seri : Modul Diskusi Fakultas Ilmu Komputer

UNIVERSITAS Narotama
A TOTAL QUALITY UNIVERSITY

FAKULTAS ILMU KOMPUTER

Sistem Komputer & Sistem Informasi

HANDOUT :

ALGORITMA
dan
PEMROGRAMAN

Oleh :

Tony Hartono Bagio

2007

KATA PENGANTAR

Algoritma dan Pemrograman adalah salah satu mata diskusi di Fakultas Ilmu Komputer Universitas Narotama, buku ini merupakan modul untuk menunjang mata kuliah tersebut, diharapkan buku ini menjadi buku pegangan bagi mahasiswa sebagai modul (hand out) untuk melengkapi isi dari modul ini, saya harap para mahasiswa membaca buku-buku lain yang sejenis, karena modul (hand out) ini sangat jauh dari sempurna.

Penulis mengerjakan modul / hand out ini setelah menggabungkan beberapa slide pada awal diskusi Algoritma & Pemrograman, versi 2007 ini mempunyai tambahan 3 bab dari versi 2003 (sampai bab XI), yaitu tambahan pada Bab XII (*Searching*), XIII (*Sorting*) dan XIV (*Matrix*), mudah-mudahan buku ini dapat berguna bagi semua mahasiswa, karena penulis merasa bahwa dalam modul ini banyak *human error* (salah mengetik / mengeja atau sejenisnya), selain itu banyak contoh soal atau soal yang harus diselesaikan/dikerjakan mahasiswa yang belum dimasukkan dalam modul ini.

Pembagian diskusi ini dibagi beberapa minggu, yang terdiri dari :

I. Pengantar Algoritma & Dasar – Dasar Algoritma
II. Aturan Penulisan Algoritma + Tipe, Nama, Nilai
III. Runtunan (*Sequence*)
IV. Pemilihan (*Selection*)
V. Pengulangan (*Looping*)
VI. Modul (latihan)
VII. Prosedur / Fungsi
VIII. UTS (Ujian Tengah Semester)
IX. Prosedur / Fungsi (lanjutan)
X. Larik (*Array*)
XI. Pencarian (*Searching*)
XII. Pencarian (*Searching*) / lanjutan
XIII. Pengurutan (*Sorting*)
XIV. Pengurutan (*Sorting*) / Lanjutan
XV. Matrix (lanjutan II)
XVI. UAS (Ujian Akhir Semester)

Angka Romawi diatas, merupakan jadwal pertemuan, termasuk jadwal Ujian, baik UTS (Ujian Tengah Semester) maupun UAS (Ujian Akhir Semester), sedang jadwal tugas tidak tercantum diatas, melainkan tergantung dari situasi.

Bila para pembaca sekalian ingin memberi tambahan, koreksi atau penyempurnaan, penulis menghaturkan terima kasih sebelumnya, karena modul ini pasti akan menjadi lebih bermanfaat.

Atas perhatiannya dalam membaca modul ini, penulis mengucapkan terima kasih.

Narotama, 15 Maret 2007

Tony Hartono Bagio
Dekan Fakultas Ilmu Komputer
Universitas Narotama Surabaya

DAFTAR ISI

BAB I
ALGORITMA

1.1 Definisi

Algoritma merupakan urutan langkah-langkah untuk menyelesaikan masalah yang disusun secara sitematis. Algoritma dibuat dengan tanpa memperhatikan bentuk yang akan digunakan sebagai implementasinya, sehingga suatu Algoritma dapat menjelaskan "*bagaimana*" cara melaksanakan fungsi yang dapat diekspresikan dengan suatu program atau suatu komponen fisik.

Untuk menyelesaikan persoalan programmer haruslah :

1) Dapat mendesain algoritma yang menjelaskan bagaimana persoalan tersebut diselesaikan.
2) Menulis / merubah algoritma menjadi suatu program dengan menggunakan suatu Bahasa Pemrograman yang sesuai.
3) Menggunakan komputer untuk menjalankan program.

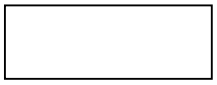
1.2 Penyajian Algoritma

Algoritma biasanya disajikan dalam dua bentuk, yaitu :

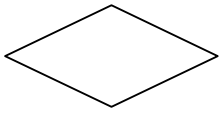
1) Menggunakan *Flow Chart (diagram alir)*
2) Menggunakan *Pseudo-Code.*

Flow-Chart

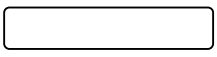
- Menggambarkan secara keseluruhan urutan proses / logika, dimana persoalan yang ada akan diselesaikan, dan menunjukkan tingkat dari detail penyelesaian persoalan
- Merupakan cara yang klasik dalam menyajikan algoritma, digunakan simbol-simbol sebagai berikut :



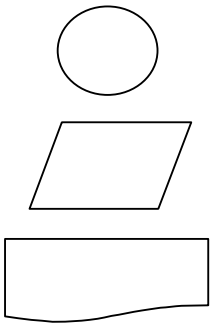
Simbol dari proses



Menyatakan keputusan (ya / tidak)



Start / Stop



Penghubung pada halaman yang sama

Pernyataan yang berhubungan dengan Input / Output

Cetak hasil

Pseudo Code

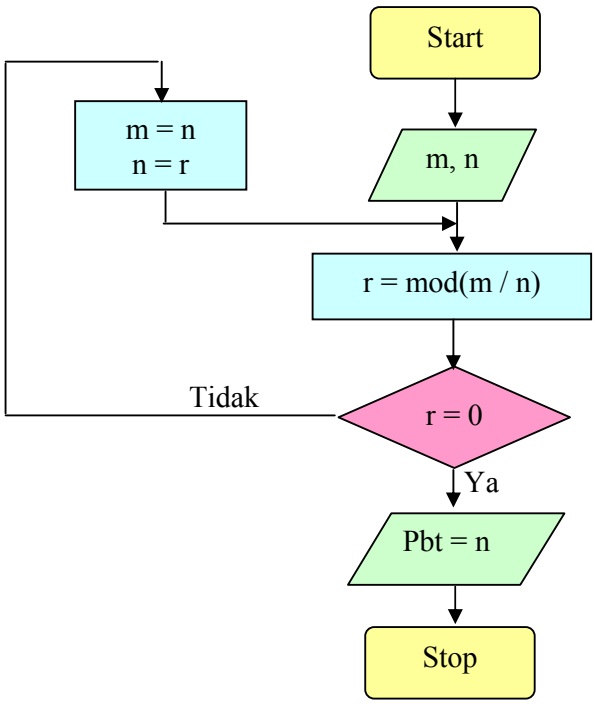
Stepwise Refinement Of Algorithms disebut juga sebagai "*top-down*" desain. Ide dari "*stepwise refinement*" adalah membagi suatu proses yang akan diselesaikan menjadi sejumlah langkah-langkah, yang masing-masing dijelaskan dengan algoritma yang lebih kecil dan lebih sederhana dari pada proses secara keseluruhan.

Contoh : ***Algoritma Euclidean***

Yaitu proses untuk menentukan pembagi bersama terbesar dari dua bilangan bulat.

Ada dua bilangan bulat ***m*** dan ***n*** (m ≥ n). Carila pembagi terbesar (pbt) dari kedua bilangan tersebut, yaitu bilangan positif terbesar yang habis dibagi m dan n.

Dengan Flow-Chart



Dengan Pseudo-Code

Deskripsi

1) Bagilah m dengan n, misalkan r adalah sisanya
2) Jika r = 0, maka n adalah jawabannya. **Stop**

   Jika r ≠ 0, lakukan langkah 3
3) Ganti nilai m dengan nilai n, dan nilai n diganti dengan nilai r, ulangi langkah 1

Contoh dengan angka:

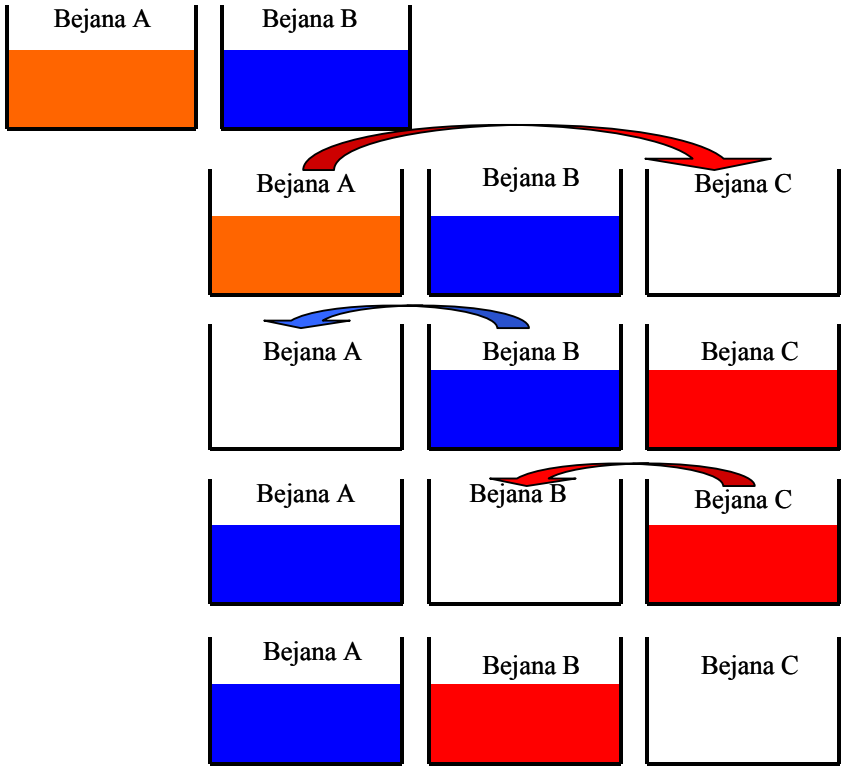
m = 30; n = 12

1.1	Hitung r = sisa (m/n)	*' r = sisa(30/12) = 6*
2.1	Check r, r ≠ 0, lakukan langkah 3	
3.1	m = n ; n = r	*'m = 12 ; n = 6*
1.2	Hitung r = sisa (m/n)	*'r = sisa(12/6) = 0*
2.2	Check r; r = 0; selesai	*'Pbt = n = 6*

Jadi Pbt(30,12) = 6

Contoh 2:

Algoritma tukar isi bejana



Keterangan :

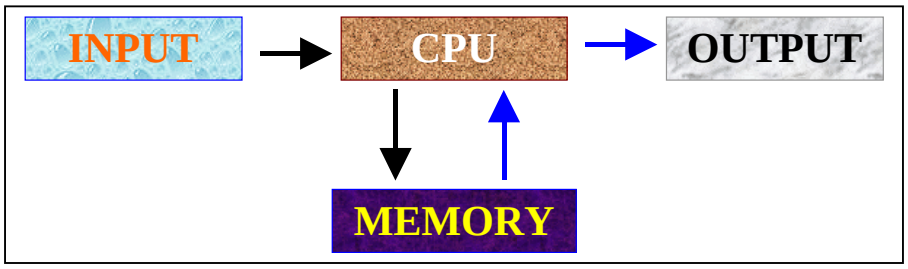
Untuk memindahkan isi dari Bejana A dan Bejana B, dibutuhkan Bejana C (sebagai tempat penampungan sementara).

1) Isi Bejana C (Kosong) dari Bejana A (warna **Merah**)
2) Isi Bejana A (setelah Kosong) dari Bejana B (warna **Biru**)
3) Isi Bejana B (setelah Kosong) dari Bejana C (warna **Merah**)
4) Maka Isi Bejana A = **Biru**; Bejana B = **Merah** dan ; Bejana C = Kosong

1.3 Algoritma Jantung Ilmu Komputer

- Algoritma tidak identik dengan Ilmu komputer saja
- Contoh Algoritma dalam kehidupan sehari-hari adalah cara membuat kue, dimana algoritma berupa suatu resep
- Setiap resep ada urutan langkah untuk membuat kue, kemudian **diproses** satu persatu setiap langkah

Pelaksanaan Algoritma dengan Pemroses



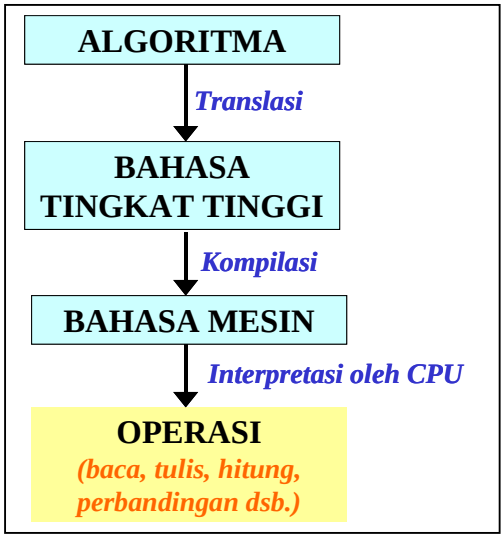
1.4 PEMROGRAMAN VS BAHASA PEMROGRAMAN

Berdasarkan Aplikasi :

- Bertujuan Khusus : Contoh COBOL *(Bisnis dan Administrasi),* FORTRAN *(Ilmiah)*, PROLOG *(Kecerdasan Buatan)*, ASEMBLER *(Pemrograman Mesin).*
- Bertujuan Umum : Contoh BASIC, PASCAL, C

Berdasarkan kedekatan:

- Bahasa tingkat Rendah : Setiap Instruksi langsung dikerjakan komputer, tanpa melalui TRANSLATOR, contoh : ASEMBLER.
- Bahasa tingkat Tinggi : Mudah dipahami , berorentasi ke manusia, perlu TRANSLATOR, contoh BASIC, PASCAL, C



1.5 Notasi Algoritma Independent terhadap Bahasa Pemrograman dan Mesin Komputer

- Notasi Algoritma diterjemahkan dalam berbagai Bahasa Pemrograman
- Bahasa Pemrograman di proses oleh Mesin Komputer.
- Analogi dengan resep kue *(Algoritma),* ditulis dengan bahasa yang berbeda *(Bahasa Pemrograman)*, dimasak oleh koki yang berbeda *(CPU)*, hasil akhir tetap sama *(Output tetap)*

Algoritma terdiri dari langkah penyelesaian masalah, jadi algoritma adalah proses *Prosedural*

Pemrograman *Prosedural* dibedakan:

- Bagian Data
- Bagian Instruksi.

Program terdiri atas *sequence* instruksi, yang dilaksanakan satu per satu secara urut oleh pemroses (CPU), instruksi dapat berubah bila ada percabangan kondisional. Data di RAM dimanipulasi oleh instruksi secara beruntun. Tahapan pelaksanaan program mengikuti pola beruntun disebut *Prosedural*

BAB II
DASAR-DASAR ALGORITMA

PROSES

Sebuah Proses dikerjakan oleh ***Pemroses (CPU)*** menurut Algoritma yang sudah ditulis

INSTRUKSI

Algoritma disusun oleh sederetan langkah ***Instruksi*** yang logis

AKSI

Tiap langkah Instruksi mengerjakan suatu tindakan (***Aksi***). Sehingga bila suatu ***Aksi*** dilaksanakan, maka sejumlah ***operasi*** yang sesuai dengan ***aksi*** dikerjakan oleh ***Pemroses***.

CONTOH PENGURUTAN DATA

1	2	3	4	5
10	15	5	17	2

No : 1 s/d 5 adalah nomor Elemen, sedangkan Angka 10, 15, 5, 17, 2 adalah data yang akan diuruntukan dari kecil ke besar.

Cari nilai terkecil dari elemen 1 sampai dengan 5 { nilai adalah 2 pada elemen ke 5}, tukarkan dengan elemen pertama (angka 10), sehingga, menjadi

1	2	3	4	5
2	15	5	17	10

Cari nilai terkecil dari elemen 2 sampai dengan 5 { nilai adalah 5, pada elemen ke 3 }, tukarkan dengan elemen ke 2 (angka 15), sehingga, menjadi

1	2	3	4	5
2	5	15	17	10

Cari nilai terkecil dari elemen 3 sampai dengan 5 { nilai adalah 10, pada elemen ke 5 }, tukarkan dengan elemen ke 3 (angka 15), sehingga, menjadi

1	2	3	4	5
2	5	10	17	15

Cari nilai terkecil dari elemen 4 sampai dengan 5 { nilai adalah 15, pada elemen ke 5 }, tukarkan dengan elemen ke 4 (angka 17), sehingga, menjadi

1	2	3	4	5
2	5	10	15	17

SELESAI

Secara Algoritma, dapat ditulis sebagai beribut :
Diberikan N data yang tersusun secara acak, uruntukan kumpulan data tersebut, sehingga
Data_ke_1 ≤ Data_ke_2 ≤ … ≤ Data_ke_N

DESKRIPSI
Untuk pass k = 1,2,… N-1, lakukan :

Data ke_k dianggap data terkecil sementara (*min*)
Untuk j = k+1, k+2,…. N, lakukan :
Jika data ke_j < *min*, maka isikan data ke_j sebagai *min* yang baru
Tempatkan data ke_k di sebuah tempat sementara (*temp*)
Tempatkan *min* ditempat data ke_k yang lama
Tempatkan *temp* ditempat min yang lama

STRUKTUR DASAR ALGORITMA

1. Urutan (*sequence*)

Terdiri dari satu atau lebih instruksi. Tiap instruksi dikerjakan urut sesuai urutan penulisannya.

2. Pemilihan (*selection*)

Instruksi dikerjakan bila telah memenuhi kondisi tertentu, contoh : if **kondisi** then **aksi1(*kondisi = true*)** else **aksi2(*kondisi = false*)**

3. Pengulangan (*repetition*)

Mengerjakan instruksi yang berulang, contoh : repeat **aksi** until **kondisi** (***true** = **selesai**; **false** = **ulang***).

BAB III
ATURAN PENULISAN ALGORITMA

3.1 TEKS ALGORITMA

b) Kepala Algoritma
c) Deklarasi
d) Deskripsi

CONTOH
MENGHITUNG_NILAI_RATA_RATA

(Menghitung nilai rata-rata sekumpulan bilangan bulat yang dibaca dari piranti masukan)

```
DEKLARASI
    X       :  integer      { peubah data bilangan bulat }
    N       :  integer      { banyaknya data bilangan bulat,  N > 0 }
    K       :  integer      { pencacah jumlah pengulangan }
    Jumlah  :  integer      { jumlah seluruh data bilangan bulat }
    Rata    :  real         { rata-rata data bilangan bulat }

DESKRIPSI :
    read (N)            { baca banyaknya data )
    k ← 1               { mulai dari data pertama }
    jumlah  ← 0
    while  k ≤  N  do
        read (X)
        jumlah ← jumlah  +  X
        k ← k+1         { cacah pengulangan selanjutnya }
    endwhile
    ( k > N )

    rata ← jumlah/N     { rata-rata data bilangan bulat }
    write (rata)
```

3.2. TRANSLASI TEKS ALGORITMA KE TEKS BAHASA PEMROGRAMAN

Program Menghitung_Nilai_Rata_Rata

```
{Menghitung nilai rata-rata sekumpulan bilangan bulat dari piranti Masukan}
(* DEKLARASI*)
var
    x   :  integer ;   {variabel bilangan bulat}
    N   :  integer ;   {banyaknya bilangan bulat, N> 0}
    k   :  integer ;   {Pencacah jumlah pengulangan}
jumlah :  integer ;
    rata:  real
```

```
(* DESKRIPSI *)

begin
        write( 'Jumlah data'); readln (N);     {baca banyaknya data }
        k := 1;             {mulai dari data pertama}
        jumlah := 0;
        while k <= N do
          begin
                write('x = ?'}; readln X;
                jumlah := jumlah + X;
                k := k + 1;
          end;

        rata := jumlah / N;
        writeln ('Rata-rata = ', rata);
end.
```

BAB IV
TIPE DATA

Tipe Data terdiri dari 4 jenis yakni :

a) Tipe Dasar
b) Tipe Bentukan
c) Nama
d) Nilai

4.1 TIPE DASAR

Bilangan Logika (*true*/*false*)

Tipe : Boolean
Domain : True/False (1/0)
Konstanta : TRUE dan FALSE
Operasi : AND; OR; NOT ; XOR

Bilangan Bulat (*integer*)

Tipe : Integer
Domain :

byte = 0 ...255 (2^8)
shortint = -128 ... 127 (2^8)
word = 0 ... 65535 (2^{16})
integer = -32768 ... 32767 (2^{16})
longint = -2147483648 ... 2147483647 (2^{32})

Konstanta : Ditulis tanpa ada titik desimal contoh: -20, 0, 9999
Operasi :

Aritmatika : + ; - ; * ; div ; mod (*tipe integer tidak mengenal " / " diganti dengan* div)
Perbandingan : < ; ≤ ; > ; ≥; = ; ≠

Bilangan Riil (*real*)

Tipe : Real
Domain (positif):

Real = 2.9×10^{-39} ... 1.7×10^{38}
single = 1.5×10^{-45} ... 3.4×10^{38}
double = 5.0×10^{-324} ... 1.7×10^{308}
single = 3.4×10^{-4932} ... 1.1×10^{4932}

Konstanta : Ditulis **harus dengan** titik desimal contoh: -28.85 , 0.0 ; 99.99 ; 7.56+E57 ; .8
Operasi :

Aritmatika : + ; - ; * ; / (real tidak mengenal div)
(Gabungan : Integer dan Real = hasil Real)

Perbandingan : < ; ≤ ; > ; ≥; =

(real tidak mengenal "≠"; karena 1/3 tidak sama dengan 0.3333)

Karakter

Tipe : char

Domain :

'a','b',...,'z','A','B', ..., 'Z','0','1',...,'9',
tanda baca, operator aritmateika ('+','-','','/'),*
karakter khusus ('$','%','@','#', dll)

Konstanta : Ditulis dengan diapit tanda petik tunggal
contoh: 'h', 'Z', '+', '9', '0', '$'
(Ingat **'9'** = char, **9** = integer **!!**)

Operasi : Perbandingan : < ; ≤ ; > ; ≥; = ; ≠

4.2 TIPE BENTUKAN

- String (kumpulan karakter, karakter adalah string dengan panjang 1)
- Tipe dasar diberi nama baru
- Record

4.2.1 STRING

Tipe : String *(Char adalah String dengan panjang 1)*

Domain : Deretan karakter *seperti pada domain karakter.*

Konstanta : Ditulis dengan diapit tanda petik tunggal
contoh: 'halo', 'SAYA', '+', '9858', '0', '$' .

Operasi :

Penyambungan (Concatenation) : +
Perbandingan : < ; ≤ ; > ; ≥; = ; ≠

4.2.2 TIPE BENTUKAN

Yakni membuat tipe dasar dengan kata kunci type

Contoh :

type Bilbulat : integer

Bila *variabel* P bertipe Bilbulat, maka *variabel* P bertipe integer

4.2.3 REKORD

Rekord/Rekaman disusun oleh satu atau lebih *field*. Tiap field menyimpan data dari tipe dasar tertentu atau tipe bentukan yang sudah didefinisikan terlebih dahulu
Contoh:

```
type DataMhs : record
                    < NIM : integer,
                      NamaMHS : string,
                      Nilai : char
                    >
```

Bila dideklarasikan M adalah variabel bertipe DataMHS, maka mengacu tiap field pada rekaman M adalah :

M.NIM ; M.NamaMHS; M.Nilai

4.3 NAMA

Variabel

Contoh :

```
DEKLARASI
x : real
k : integer
```

x dan k adalah variabel yang **dapat** diubah didalam algoritma

Konstanta (*constant*)

Contoh :

```
DEKLARASI
const phi = 3.14
const maks = 100
```

phi dan maks adalah konstanta yang **tak dapat** diubah didalam algoritma

Tipe bentukan

Contoh:

```
DEKLARASI
    type JAM : record
                    < hh   : integer,
                      mm   : integer,
                      ss   : integer
                    >
    J1, J2 : JAM
```

J1, J2 adalah nama variabel yang bertipe JAM

Fungsi (*function*) dan Prosedur (*procedure*)

Contoh:

DEKLARASI

function MAKSIMUM(A,B : integer) → integer

procedure TUKAR(input/ouput A,B: integer)

MAKSIMUM adalah nama Fungsi

TUKAR adalah nama Prosedur

4.4 NILAI

Pengisian Nilai ke *variabel*

- Secara langsung (*assignment*): var_a ← b
- Dari piranti masukan : read (var_1, var_2, …, var_n)

Ekspresi

- Aritmatika : var_c ← var_a * var_b
- Relasional : x < 6 (hasil *true*)
- String : a + ‘c’ (dimana a bertipe *string*)

Menuliskan Nilai ke piranti *output*

- write (var_1, var_2,…..var_n)

CONTOH-CONTOH MENDEFINISIKAN NAMA DI DALAM BAGIAN DEKLARASI

```
DEKLARASI
    { nama tetapan )
    const phi = 3.14            { tetapan  π }
    const Nmaks = 100           { jumlah mahasiswa }
    const sandi = ‘xyz’         { kunci rahasia }

    { nama tipe }

    type MHS : record           { data mahasiswa }
                 < NIM : integer,
                   nama : string,
                   usia : integer
                 >

    type Titik      : record <x:real, y:real>     { titik didalam bidang kartesian }
    type Kompleks   : record <a:real, b:real>     { bilangan kompleks }

    tipe JAM : record
                  < hh   : integer,  { 0 . . . 23 }
                    mm   : integer,  { 0 . . . 59 }
                    ss   : integer,  { 0 . . . 59 }
                  >
```

```
tipe JADWAL_KA : record
                 <  NoKA         : string,
                    KotaAsal     : string,
                    JamBerangkat : JAM,
                    KotaTujuan   : string,
                    JamTiba      : JAM
                 >
{ nama peubah/variabel }
luasL   :  real       { luas lingkaran }
psw     :  string     { password }
indeks  :  char       { indeks nilai ujian }
ketemu  :  boolean    { hasil pencarian, true jika data yang dicari ditemukan,
                        false jika sebaliknya)
P       :  Titik      { koordinat titik pusat lingkaran }
F       :  Kompleks   { frekuensi, hasil transformasi Fourier }
JKA     :  JadwalKA   { jadwal kereta api ekspres }
```

{ namafungsi dan prosedur }

```
Procedure HITUNG_TITIK_TENGAH ( input P1:Titik, input P2:Titik, output
Pt:Titik )
( menghitung titik tengah garis dengan titik ujung P1 dan P2)

function FAKTORIAL (n: integer) → integer
{ mengembalikan nilai faktorial dari n > 0 }

function CARI (input x: integer) → boolean
{ true bila x ditemukan, false bila sebaliknya }

procedure CETAK_JADWAL_KA ( input kota : string)
( mencetak jadwal semua kereta api yang berangkat dari kota tertentu)
```

CONTOH CETAK HALLO

Algoritma CETAK_HALO_1

```
{ Mencetak string 'Halo, dunia' ke piranti keluaran }

DEKLARASI
    { tidak ada }

DESKRIPSI
    write ('Halo, dunia')
```

Algoritma CETAK_HALO_2

```
{ Mencetak string 'Halo, dunia' ke piranti keluaran }

DEKLARASI
    ucapan : string

DESKRIPSI
```

```
    ucapan ← 'halo, dunia'
    write (ucapan)
```

Algoritma CETAK_HALO_3
{ Mencetak string 'HALO, dunia' ke piranti keluaran }

```
DEKLARASI
    const ucapan : 'HALO'

DESKRIPSI
    write (ucapan)
```

Algoritma CETAK_HALO_4
{ Mencetak string 'Halo' dan diikuti dengan nama orang.}
{ Nama orang diaca dari piranti masukan }

```
DEKLARASI
    nama_orang : string

DESKRIPSI
    read (nama_orang)
    write ('Halo', nama_orang)
```

ALGORITMA

```
DEKLARASI
    type Titik    : record <x:real, y:real>
    P             : Titik
    a, b          : integer
    NamaArsip     : string
    Nilai         : real
    C             : char

DESKRIPSI
    nilai ← 1200.0
    read (p.x, p.y)
    read (NamaArsip)
    read (a, b)
    read (C)

    write ( 'Nama arsip: ',NamaArsip)
    write ( 'koordinat titik adalah: ',p.x, ', ', p.y )
    write ( b, nilai)
    write ( 'karakter yang dibaca adalah ', C )
```

BAB V
URUTAN

5.1 PENGERJAAN URUTAN

1) Instruksi dikerjakan satu per satu
2) Instruksi dikerjakan sekali, tidak ada instruksi yang diulang
3) Urutan instruksi yang dilaksanakan pemroses = urutan aksi dalam Algoritma
4) Akhir instruksi = akhir Algoritma

Contoh:

A1

A2

A3

A4

Mula-mula A1 dikerjakan; setelah selesai , mengerjakan A2, setelah itu A3, dilanjuntukan A4, tanpa ada yang mengulang

5.2 PENGARUH URUTAN INSTRUKSI

Algoritma Urutan1

```
DEKLARASI
        A, B : integer
DESKRIPSI
        A ← 10              { A = 10 }
        A ← A * 2           { A = 10 *2=20 }
        B ← A               { B = 20 }
        write (B)
```

*Hasil **B** pada Output =* **20**

```
Algoritma Urutan2
DEKLARASI
        A, B : integer
DESKRIPSI
        A ←10               { A = 10 }
        B ← A               { B = 10 }
        A ← A * 2           { A = 10*2=20}
        write (B)
```

*Hasil **B** pada Output =* **10**

Algoritma Swap

```
{menukar tempat A dan B, dari piranti INPUT }
DEKLARASI
        A, B, C  : integer  DESKRIPSI     {baca nilai A dan B }
        read (A, B)                       {tulis/cetak nilai A dan B sebelum ditukar }
        write (A, B)                      { proses penukaran }
```

```
C ← A          { simpan nilai A, pada variable sementara C }
A ← B          { Isi Variabel A dengan nilai B }
B ← C          { Isi Variabel B dengan nilai A, yang tersimpan di C }
               {tulis/cetak nilai A dan B sesudah ditukar }
write (A, B)
```

Contoh 5.1
MENGHITUNG LUAS SEGI TIGA

Menghitung luas segitiga. Luas sembarang segitiga adalah setengah dari panjang alas dikali tinggi. Panjang alas dan tinggi segitiga dibaca dari piranti masukan. Luas segitiga dicetak ke piranti keluaran.

Algoritma LUAS_SEGITIGA

{ Dibaca panjang alas (a) dan tinggi (t) segitiga. Hitunglah luas segitiga tersebut. Untuk panjang alas dan tinggi tertentu. Luas segitiga dihitung dengan rumus L = ½ at. Nilai L dicetak ke piranti keluaran }

```
DEKLARASI
    a : real   { panjang alas segitiga, dalam satuan cm }
    t : real   { tinggi segitiga, dalam satuan cm }
    L : real   { luas segitiga, dalam satuan cm }

DESKRIPSI
    read (a)
    read (t)
    L ← a * t /2
    write (L)
```

CONTOH 5.2
KOMISI SALESMAN

Menghitung komisi yang diterima *salesman* berdasarkan jumlah-jumlah penjualan yang dicapainya. *Salesman* itu mendapatkan komisi *10%* dari hasil penjualannya. Masukan algoritma adalah nama salesman dan jumlah yang dicapainya. Tampilkan piranti keluaran nama *salesman* dan besar komisi yang diperolehnya.

Algoritma KOMISI_SALESMAN

{ Menghitung komisi yang diterima seorang salesman. Besar komisi adalah 10% dari nilai penjualan yang dicapainya.}
{ Data masukan adalah nama salesman dan nilai penjualannya. Keluaran algoritma adalah besar komisi yang diterima salesman tersebut }

```
DEKLARASI
    nama_salesman   : string
    nilai_penjualan : real   { nilai penjualan yang dicapai, dalam Rp }
    komisi          : string { besar komisi, dalam Rp }

DESKRIPSI
```

```
read ( nama_salesman, nilai_penjualan )
komisi ← 0.1 * nilai_penjualan
write ( komisi )
```

Contoh 5.3

Dibaca nama karyawan dan gaji pokok bulanannya. Gaji yang diterima pegawai adalah :

Gaji bersih = gaji pokok + tunjangan – pajak

Tunjangan karyawan dihitung 20% dari gaji pokok, sedangkan pajak adalah 15% dari gaji pokok ditambah tunjangan.

```
Algoritma GAJI_KARYAWAN
{Menghitung gaji bersih karyawan. Data masukan adalah nama karyawan dan gaji
pokok bulanannya. Gaji bersih = gaji pokok + tunjangan – pajak.
Tunjangan adalah 20% dari gaji pokok. Sedangkan pajak adalah 15% dari gaji pokok +
tunjangan.}

DEKLARASI
    nama_karyawan      : string
    gaji_pokok, tunjangan, pajak, gaji_bersih  :   real

DESKRIPSI
    read ( nama_karyawan, gaji_pokok )
    tunjangan ← 0.2 * gaji_pokok
    pajak ← 0.15 *  (gaji_pokok + tunjangan)
    gaji_bersih ← gaji_pokok + tunjangan - pajak
    write ( gaji_bersih )
```

CONTOH 5.4

Dibaca dua buah titik P1(x_1, y_1) dan P2(x_2, y_2).Tulislah dari P1 dan P2

Titik tengah dari dua P1 dan P2 adalah P3(x_3, y_3) yang dihitung dengan rumus :

$$x_3 = \frac{x_1 + x_2}{2} \cdot dan \cdot y_3 = \frac{y_1 + y_2}{2}$$

Sebagai contoh, P1(4,8) dan P2(2,6), maka P3(3,7)

Algoritma TITIK_TENGAH

```
{ Dibaca dua buah titik P1(x1, y1) dan P2(x2, y2). Hitunglah titik tengah daari kedua
buah titik tersebut. }

DEKLARASI
    type Titik      :      record <      x: real,
                                                y: real
                                                >
    P1, P2, P3 : Titik
```

```
DESKRIPSI
        read (P1.x, P1.y)                   { baca titik P1 }
        read (P2.x, P2.y)                   { baca titik P2 }
        P3.x ← (P1.x + P2.x) / 2
        P3.y ← (P1.y + P2.y) / 2
        write ( P3.x, P3.y )
```

CONTOH 5.5

Mengkonversi konversi jam-menit-detik (hh:mm:ss) menjadi total detik

Data jam-menit-detik dibaca dari piranti masukan

Ingat : 1 menit = 60 detik

1 jam = 3600 detik

Misalnya, 1 jam, 5 menit, 40 detik adalah (1 x 3600) + (5 x 60) + 40 = 3940 detik

Algoritma KONVERSI_JAM_MENIT_DETIK

{ Dibaca jam-menit-detik (hh:mm:ss) Konversilah jam-menit-detik ke dalam total detik }.

```
DEKLARASI
    type Jam   :   record <hh  : integer,      {jam = 0...23}
                           mm : integer,       {menit  = 0..59}
                           ss   : integer      {detik = 0...59}
                          >
    J  :  Jam
    Total_detik  :  integer

DESKRIPSI
    read (J.hh, J.mm, J.ss)
    total_detik ← (J.hh * 3600) + (J.dd * 60) + J.ss
    write (total_detik)
```

CONTOH 5.6

Dibaca lama sebuah percakapan telepon dalam satuan detik. Tulislah algoritma untuk menghitung berapa jam, berapa menit, dan berapa detikkah lama percakapan tersebut.

Contoh : 4000 detik = 1 jam + 6 menit + 40 detik, ini diperoleh dengan cara:

```
4000   div   3600   =   1     (jam)
4000   mod   3600   =   400   (sisa detik)
400    div   60     =   6     (menit)
400    mod   60     =   40    (detik)
```

Algoritma KONVERSI_TOTAL_DETIK_KE_JAM
{ Mengkonversi jumlah detik menjadi jam-menit-detik }.

```
DEKLARASI
    type Jam   :   record <hh  : integer,      {jam = 0...23}
                           mm : integer,       {menit  = 0..59}
                           ss   : integer      {detik  = 0..59}
                          >
    J  :  Jam
    total_detik  :  integer
    sisa :  integer    {peubah / variabel pembantu}

DESKRIPSI
    read (total_detik)
    J.hh ← total_detik  div   3600    { mendapatkan jam }
    Sisa ← total_detik  mod  3600
    J.mm ← sisa div   60              { mendapatkan menit }
    J.ss   ← sisa mod  60             { mendapatkan detik }
    write (J.hh, J.mm, J.ss)
```

CONTOH 5.7

Menghitung selisih waktu dari dua buah jam, J1 (hh:mm:ss) dan jam J2 (hh:mm:ss) dengan syarat J2.hh > J1.hh.

Misalnya,

J2	J1	J3 = J2 – J1
12:10:56	10:08:14	02:02:42
12:18:40	10:12:50	02:05:50
12:10:40	10:40:55	01:29:45

Algoritma SELISIH_JAM
{ Dibaca dua buah jam, J1 dan J2, Hitunglah selisih J2 - J1 = J3 }.

```
DEKLARASI
    type Jam   :   record < hh  : integer,      {jam = 0...23}
                           mm : integer,       {menit  = 0..59}
                           ss   : integer      {detik  = 0..59}
                          >
    J1, J2, J3   :  Jam
    total_detik1, total_detik2, selisih_detik  :  integer
    sisa :  integer    {peubah/variabel pembantu}

DESKRIPSI
    read (J1.hh, J1.mm, J1.ss)        { jam pertama }
    read (J2.hh, J2.mm, J2.ss)        { jam kedua }

    { konversi jam ke total_detik }
```

```
total_detik1 ← (J1.hh * 3600) + (J1.mm * 60) + J1.ss
total_detik2 ← (J2.hh * 3600) + (J2.mm * 60) + J2.ss

{ hitung selisih total detik }
selisih_detik ← total_detik2 – total_detik1

{ konversi selisih_detik ke dalam jam-menit-detik }
J3.hh ← selisih_detik div 3600          { mendapatkan jam }
sisa  ← selisih_detik mod 3600
J3.mm ← sisa div 60                     { mendapatkan menit }
J3.ss ← sisa mod 60                     { mendapatkan detik }

write (J3.hh, J3.mm, J3.ss)
```

CONTOH 5.8

Misalkan seorang penelpon di warung telekomunikasi memulai percakapan pada pukul J1 dan selesai pada pukul J2. Bila 1 pulsa = 5 detik, dan biaya per pulsa Rp. 150.
Tulislah algoritma untuk menghitung lama percakapan (dalam jam-menit-detik) dan biaya yang harus dibayar penelpon.. Untuk menyederhanakan masalah, andaikanlah wartel itu tutup tepat pada pukul 00:00:00 dini hari

Algoritma WARTEL

```
{ Menghitung biaya percakapan di warung telekomunikasi (wartel). Biaya percakapan
dihitung dari lima percakapan }.

DEKLARASI
    const biaya_per_pulsa = 150          { biaya per pulsa }
    type Jam  :  record < hh : integer,  {jam = 0...23}
                          mm : integer,  {menit  = 0..59}
                          ss : integer   {detik  = 0..59}
                        >

    J1 : Jam      { jam awal percakapan }
    J2 : Jam      { jam selesai percakapan }
    J3 : Jam      { lama percakapan }

    total_detik1, total_detik2 : integer
    sisa   : integer   {peubah/variabel pembantu}
    lama   : integer
    pulsa  : real
    biaya  : real

DESKRIPSI
    read (J1.hh, J1.mm, J1.ss)      { jam awal percakapan }
    read (J2.hh, J2.mm, J2.ss)      { jam selesai percakapan }

    { konversi jam ke jumlah_detik }
```

```
total_detik1 ← (J1.hh * 3600) + (J1.mm * 60) + J1.ss
total_detik2 ← (J2.hh * 3600) + (J2.mm * 60) + J2.ss

{ hitung lama percakapan dalam jumlah detik }
lama ← total_detik2 – total_detik1

{ hitung jumlah pulsa dan biaya untuk seluruh pulsa }
pulsa ← lama / 5
biaya ← pulsa * biaya_per_pulsa

{ konversi selisih_detik ke dalam jam-menit-detik }
J3.hh ← lama  div   3600      { mendapatkan jam }
sisa  ← lama  mod  3600
J3.mm ← sisa  div   60        { mendapatkan menit }
J3.ss  ← sisa  mod  60        { mendapatkan detik }

write (J3.hh, J3.mm, J3.ss, biaya)
```

CONTOH 5.9

Bagaimana cara mempertukarkan nilai A dan B ?
Misalnya, sebelum pertukaran nilai A = 8, nilai B = 5, maka setelah pertukaran, nilai A = 5 dan nilai B = 8, andaikan nilai yang dipertukarkan bertipe integer

Masalah ini sama dengan mempertukarkan isi dua buah bejana (lihat Bab 1).
Kalau anda menulis algoritmanya seperti ini :

```
A ← B
B ← A
```

Maka hasilnya A = 5 dan B = 8.
Jadi, algoritma pertukarannya salah.
Dalam mempertukarkan nilai dua buah peubah, perlu digunakan peubah/variabel bantu, misalnya C.

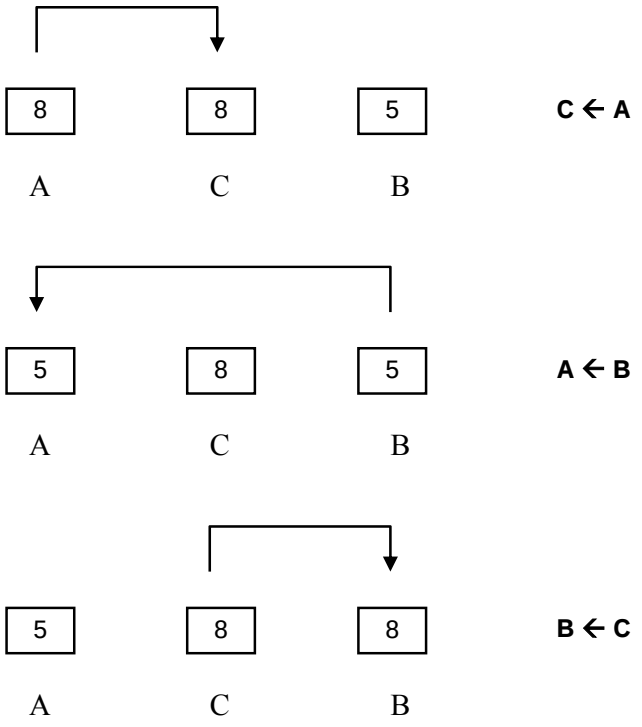
```
C ← A        { simpan nilai A di tempat penampungan, C }
A ← B        { sekarang nilai A  dapat diisi dengan  nilai B }
B ← C        { isi B  dengan nilai A  semula yang tadi disimpan di  C }
```

Ketiga instruksi penugasan ini dapat ditunjukkan pada sebagai berikut :

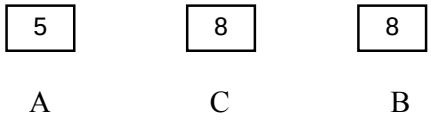
Sebelum pertukaran :

A C B

Proses pertukaran :



Setelah pertukaran :



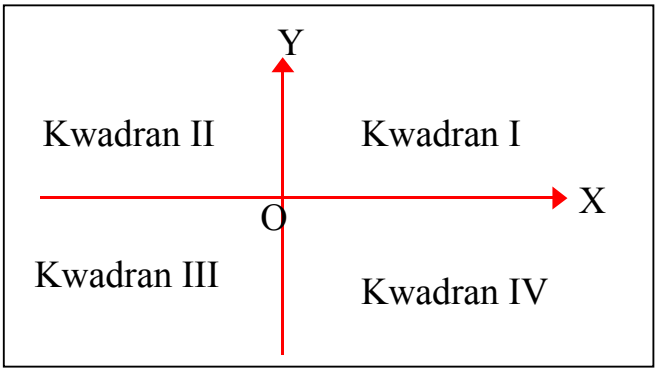
Gambar 5.2 Proses mempertukarkan nilai A dan B dengan menggunakan peubah bantu C. Peubah bantu C berguna untuk menyimpan sementara nilai peubah A. Perhatikanlah, bahwa setelah pertukaran A dan B, peubah C masih berisi nilai peubah semula (8). Hal ini karena pengisian nilai C←A sama dengan menyalin (copy) nilai peubah A ke dalam peubah C. Peubah A sendiri tidak kehilangan nilai akibat pengisian nilai tersebut. Sehingga pengisian nilai B←C tetap meninggalkan nilai di dalam peubah C.

Algoritma TUKAR
{ Mempertukarkan nilai A dan B. Nilai A dan B dibaca dari piranti masukan }.

```
DEKLARASI
    A : integer   { nilai pertama }
    B : integer   { nilai kedua }
    C : integer   { peubah/variable pembantu }

DESKRIPSI
    { baca nilai A dan B }
    read ( A, B )

    { cetak nilai A dan B sebelum pertukaran }
    write ( A, B )

    { proses pertukaran }
    C ← A      { simpan nilai A di tempat penampungan sementara, C }
    A ← B      { sekarang A dapat diisi dengan nilai B }
    B ← C      { isi B dengan nilai A semula yang tadi disimpan di C }

    { cetak nilai A dan B setelah pertukaran }
    write ( A B )
```

BAB VI
PEMILIHAN

Suatu aksi dilakukan bila telah memenuhi syarat (kondisi *true* ataupun *false*)
Kemungkinan ada suatu kondisi tertentu.
Contoh :



Pada kasus diatas ada beberapa kemungkinan untuk titik P(x,y) :

Kasus 1 : x > 0 and y > 0 ; P(x,y) di kw I
Kasus 2 : x < 0 and y > 0 ; P(x,y) di kw II
Kasus 3 : x < 0 and y < 0 ; P(x,y) di kw III
Kasus 4 : x > 0 and y < 0 ; P(x,y) di kw IV
Kasus 5 : x = 0 and y < 0 or y > 0 ; P(x,y) di sumbu y
Kasus 6 : y = 0 and x < 0 or x > 0 ; P(x,y) di sumbu x
Kasus 7 : x = 0 and y = 0 ; P(x,y) di titik O

Macam-macam Kondisi

- Satu kasus (kondisi = *true*)
- Dua kasus (kondisi = *true* or *false*)
- Lebih dari dua kasus (*nested* = percabangan)

Contoh Satu kasus

```
if  x > 100 then
    x ← x  + 1 { x > 100 kondisi ini true , maka kerjakan x = x + 1 }
endif
```

Contoh Dua kasus

```
if  x > 100 then
        x ← x  + 1      { x > 100 kondisi ini true , maka kerjakan x = x + 1 }
else
        x ← 100 - x     { x > 100 kondisi ini false , maka kerjakan x = 100 - x }
endif
```

Lebih dari Dua Kasus

```
if <kondisi1> then
      aksi1
else    if <kondisi2>  then
           aksi2
         else    if <kondisi3>  then
                        aksi3
                 else    if <kondisi4>  then
                              aksi4
                         endif
                 endif
        endif
endif
```

Dan seterusnya.

STRUKTUR CASE

Untuk kasus lebih dari dua kasus, struktur CASE dapat menyederhanakan kasus IF-THEN-ELSE yang bertingkat-tingkat

Struktur CASE adalah sebagai berikut :

```
case (nama)
        <kondisi1> : aksi1
        <kondisi2> : aksi2
        <kondisi3> : aksi3
        …
        …
        <kondisiN> : aksiN
        [otherwise  aksiX]      { optional }
endcase
```

PERBEDAAN IF-THEN-ELSE LEBIH DARI DUA KASUS DENGAN CASE

IF-THEN-ELSE

```
Algoritma WUJUD_AIR{wujud cair : padat ; cair ; uap }
DEKLARASI
        T : real        {suhu air dalam oC }
        W: string       {wujud air }
DESKRIPSI
        read (T)
        if T ≤ 0 then
           W ← 'padat'
        else
           if  T>0 and  T<100 then
                  W ← 'cair'
           else
                  W ← 'uap'
           endif
    endif
        write (W)
```

CASE

```
Algoritma WUJUD_AIR{wujud cair : padat ; cair ; uap }
DEKLARASI
        T : real {suhu air dalam oC }
        W: string {wujud air }
DESKRIPSI
        read (T)
        case (T)
           T ≤ 0          : W ← 'padat'
           0 < T<100   : W ← 'cair'
           otherwise      W ← 'uap'
        endcase
        write (W)
```

BAB VII
PENGULANGAN (*looping*)

7.1 BAGIAN STRUKTUR PENGULANG

Struktur Pengulangan terdiri dari :

- Kondisi pengulangan, apabila ekpresi *boolean* terpenuhi,
- Body pengulangan, yaitu satu atau lebih aksi yang akan diulang

Bagiab Struktur Pengulangan :

- Inisialisasi, aksi dilakukan sebelum pengulangan dilakukan satu kali
- Terminasi, aksi dilakukan setelah pengulangan selesai dilaksanakan.

7.2 JENIS STRUKTUR PENGULANGAN

- Struktur WHILE-DO
- Struktur REPEAT-UNTIL
- Struktur FOR

Pemilihan struktur apabila Kondisi tak dapat ditentukan, gunakan WHILE-DO atau REPEAT-UNTIL, sedangkan Kondisi yang sudah ditentukan, gunakan FOR

7.2.1 Struktur WHILE-DO

Bentuk Umum

```
while <kondisi> do
        aksi
endwhile
```

Aksi dilakukan berulang kali selama <kondisi> *boolean* tetap bernilai *true*. Jika <kondisi> bernilai *false*, badan pengulangan tidak dilaksanakan. Pengulangan SELESAI

7.2.2 Struktur REPEAT-UNTIL

Bentuk Umum

```
repeat
        aksi
until <kondisi>
```

Aksi didalam badan kalang diulang sampai kondisi berhenti bernilai *true*, atau dengan kata lain kondisi berhenti bernilai *false*, pengulangan dilakukan

7.2.3 WHILE-DO vs REPEAT-UNTIL

WHILE-DO

Check kondisi dilakukan pada awal looping

Pengulangan dilaksanakan min 0 x, hal ini tjd bila check kondisi pertama kali bernilai false

REPEAT-UNTIL

Check kondisi dilakukan pada akhir *looping*

Pengulangan dilaksanakan *min* 1 x, bila check kondisi pertama kali diakhir pengulangan bernilai *true*

7.2.4 STRUKTUR FOR

Umum (*ascending*)

```
for Var = St to Fi do
   aksi
endfor
```

- ***Ascending***

```
for Var = St to Fi do
   aksi
endfor
```

Var bertipe sederhana kecuali tipe *real*
St ≤ Fi; if St > Fi, then badan looping tidak dimasuki
Pada awalnya, Var = St, otomatis bertambah satu setiap kali Aksi dimasuki, sampai akhirnya Var = Fi
Jumlah looping = Fi – St +1

- ***Descending***

```
for Var = Fi downto St do
   aksi
endfor
```

Var bertipe sederhana kecuali tipe *real*
St ≤ Fi; if St > Fi, then badan looping tidak dimasuki
Pada awalnya, Var = Fi, otomatis berkurang satu setiap kali Aksi dimasuki, sampai akhirnya Var = St
Jumlah looping = Fi – St +1

7.3 CONTOH STRUKTUR

7.3.1 Contoh WHILE-DO

```
Algoritma JUMLAH_DERET          { menjumlahkan deret 1 + 2 + 3 + ..... + N }

DEKLARASI
        N, Angka, Jumlah : Integer

DESKRIPSI
        read (N)                { banyak suku deret }
        Jumlah ← 0              { inisialisasi jumlah deret }
        Angka ← 1               { suku deret }
        while angka ≤ N do
                jumlah ← jumlah + angka { jumlah deret sekarang }
                Angka ← Angka + 1       { suku deret berikutnya }
        endwhile                { angka > N ; kondisi setelah looping berhenti }
        write (jumlah)
```

7.3.2 CONTOH REPEAT-UNTIL

```
Algoritma JUMLAH_DERET          { menjumlahkan deret 1 + 2 + 3 + ..... + N }

DEKLARASI
        N, Angka, Jumlah : Integer

DESKRIPSI
        read (N)                { banyak suku deret }
        Jumlah ← 0              { inisialisasi jumlah deret }
        Angka ← 1               { suku deret }
        repeat
                jumlah ← jumlah + angka { jumlah deret sekarang }
                Angka ← Angka + 1       { suku deret berikutnya }
        until angka > N         { angka > N ; kondisi setelah looping berhenti }
        write (jumlah)
```

7.3.4 CONTOH FOR

```
Algoritma JUMLAH_DERET          { menjumlahkan deret 1 + 2 + 3 + ..... + N }

DEKLARASI
        N, Angka, Jumlah : Integer

DESKRIPSI
        read (N)                { banyak suku deret }
        Jumlah ← 0              { inisialisasi jumlah deret }
        for Angka ← 1 to N
                jumlah ← jumlah + angka { jumlah deret sekarang }
        endfor

        write (jumlah)
```

7.4 CONTOH *PENGULANGAN*

7.4.1 CONTOH PENJUMLAHAN DERET 1/x

Misal akan membuat algoritma untuk menjumlahkan deret

$$S = \frac{1}{x_1} + \frac{1}{x_2} + \frac{1}{x_3} + ...$$

{Menjumlahkan deret $1/x_1+1/x_2+1/x_3+$ ……, dengan x_1, x_2, x_3 ……. Adalah bilangan bulat yang dibaca dari piranti masukan dengan syarat $x_1 \neq 0$. Jumlah deret dicetak dipiranti keluaran}

Algoritma Jumlah_Deret_WHILE-DO

```
DEKLARASI
x : integer  {harga bil.bulat yang dibaca}
s : real     {jumlah deret}

DESKRIPSI
  read(x)
  if x = 0 then
     write('x tak boleh bernilai 0')
  else
    {inisialisasi jumlah deret}
    s ← 0
    while x ≠ 0 do
      s ← s + 1/x
    read(x)
    endwhile
    {x = 0 ; kondisi setelah pengulangan}
    write(s)
endif
```

Algoritma Jumlah_Deret_Repeat-Until

```
DEKLARASI
  x : integer  {harga bil bulat yg dibaca}
  s : real     {jumlah deret}

DESKRIPSI
  read(x)
  if x = 0 then
     write('x tak boleh bernilai 0')
  else
    {inisialisasi jumlah deret}
    s ← 0
    repeat
      s ← s + 1/x
      read(x)
    until x = 0
    {x = 0 ; kondisi setelah pengulangan}
    write(s)
  endif
```

Pada contoh diatas penggunaan WHILE-DO **tidak tepat** (walaupun hasilnya sama), karena pemeriksaan x = 0 dilakukan **dua** kali, yakni pada saat kondisi if dan while

7.4.2 CONTOH JUMLAH DATA

{Menjumlahkan sekumpulan data Bil bulat positif yang dibaca pada piranti masukan}

Algoritma Jumlah_Data(While-Do)

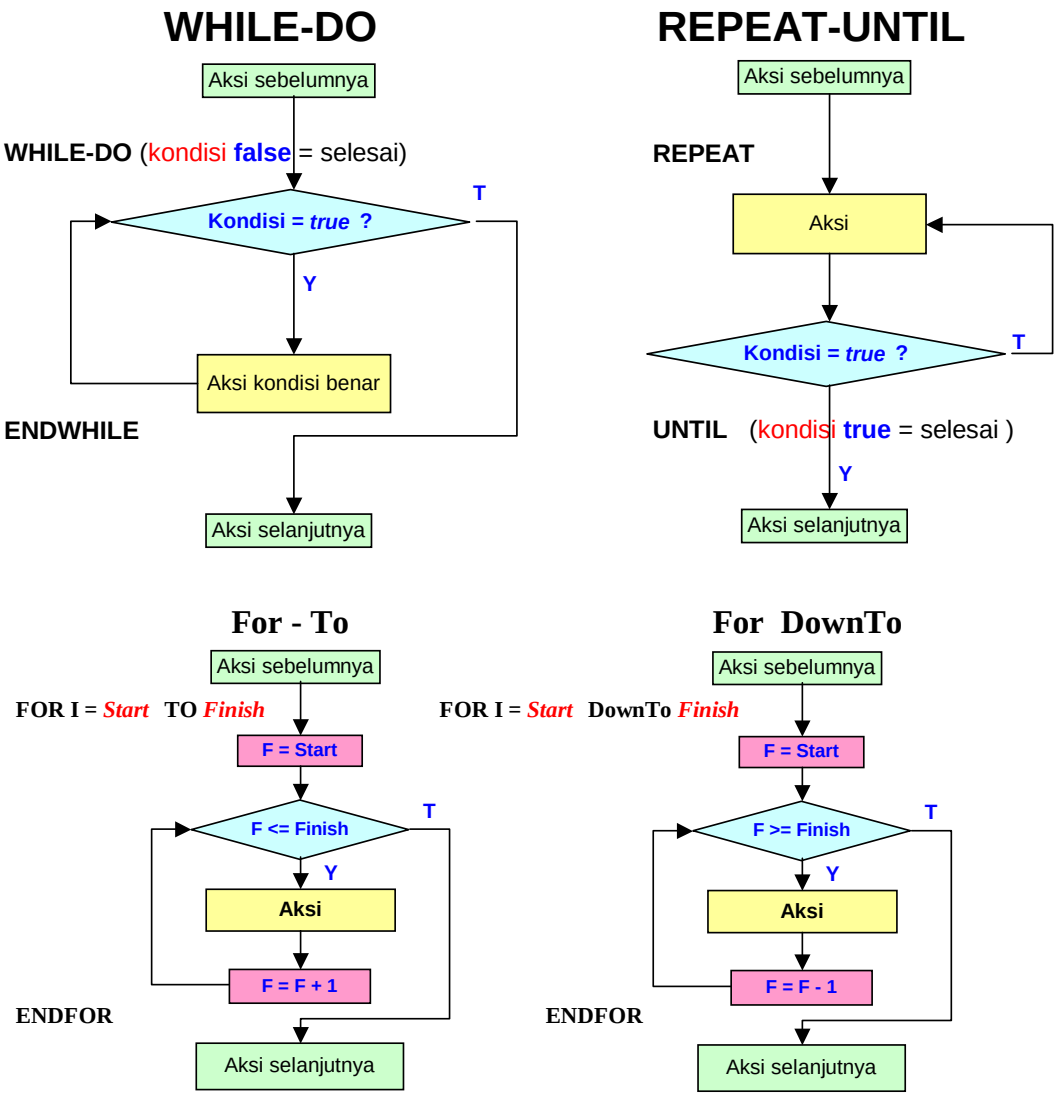
```
DEKLARASI
  bil : integer
  jumlah : integer

DESKRIPSI
  jumlah ← 0
  read(bil)
  while bil ≠ -99 do
    jumlah ← jumlah + bil
    read(bil)
  endwhile      {bil = -99}
write(jumlah)
```

Algoritma Jumlah_Data(Repeat-Until)

```
DEKLARASI
  bil : integer
  jumlah : integer

DESKRIPSI
  jumlah ← 0
  repeat bil ≠ -99
     read(bil)
     jumlah ← jumlah + bil
  until bil = -99          {bil = -99}

write(jumlah)
```

Untuk contoh diatas pemakaian REPEAT-UNTIL, tidak tepat, karena bila ada data : 20, 14, 10, -99, maka outputnya –55, bila datanya nilai –99, outputnya adalah –99. Kesimpulan diatas adalah **SALAH**, karena –99 ikut terjumlahkan.

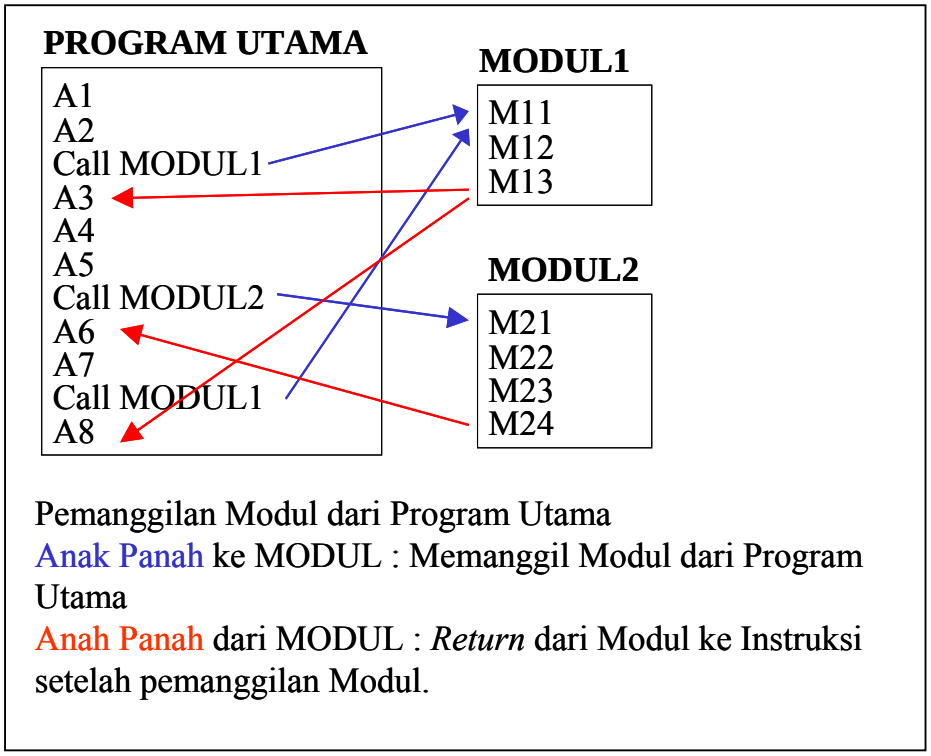
Sedangkan dengan WHILE-DO, dengan data yang sama outputnya 44, sedang yang kedua outputnya 0



BAB VIII
MODUL

8.1 PEMECAHAN PROGRAM

Program besar dipecah menjadi beberapa sub-program yang kecil, Tiap sub-program kadangkala independen dengan Program Utama. Tiap sub-program disebut MODUL Sehingga suatu program utama dapat dibuat menjadi beberapa modul. Bahasa pemrograman menamakan modul: *routine*, *procedure* atau *function*. Dan Modul dapat dipanggil(*called*) dari program utama



Pemanggilan Modul dari Program Utama
Anak Panah ke MODUL : Memanggil Modul dari Program Utama
Anah Panah dari MODUL : *Return* dari Modul ke Instruksi setelah pemanggilan Modul.

8.2 STRUKTUR MODUL

Mempunyai bentuk yang sama dengan Struktur Algoritma, yang mempunyai :

- Judul (*header*), berisi nama Modul, komentar
- Deklarasi,
- Deskripsi (*body* program), berisi instruksi yang akan menjalankan setiap aksi

Sedang Jenis MODUL ada dua macam yakni :

- PROCEDURE
- FUNCTION

CONTOH

{hanya bagian Algoritma yang sama ditulis berulang-ulang yang ditampilkan, sedang bagian Algoritma yang lain ditulis "…"}

```
ALGORITMA ABCD

DEKLARASI
        A, B, C, D,  temp : integer

DESKRIPSI
        ….
        {menukar nilai A dan B}
        temp ← A
        A ← B
        B ← temp
        …
        …
        if C > D then
                {menukar nilai C dan D}
                temp ← C
                C ← D
                D ← temp
        endif
        ….
        ….
```

Contoh Pemanggilan MODUL

```
MODUL TUKAR

Procedure TUKAR (input/output  P, Q : integer)
{menukar nilai P dan Q}

DEKLARASI
        temp : integer

DESKRIPSI
        temp ← P
        P ← Q
        Q ← temp


PROGRAM UTAMA

Algoritma ABCD
{contoh memanggil prosedur TUKAR dari program utama}

DEKLARASI
        A, B, C, D : integer
        ….
```

```
Procedure TUKAR (input /    output  P, Q : integer)

DESKRIPSI
    ….
    TUKAR(A,B)
    ….
    ….
    if C  > D then
            TUKAR(C,D)
    endif
    ….
    ….
```

8.3 KUMPULAN STUDI KASUS

STUDI KASUS 1

Didefinisikan tipe JAM dan peubah J dan p sebagai berikut :

```
Type JAM : record
                <hh    : integer;      {0 . . 23}
                  mm   : integer;      {0 . . 59}
                  ss   : integer;      {0 . . 59}
                >
J  : JAM
P : integer
```

Tulislah algoritma yang :

- Mengisi (assignment) peubah J dengan jam 16:10:34;
- Membaca p menit dari piranti masukan;
- Mengubah nilai J setelah p menit (algoritma melakukan kalkulasi; tidak boleh menggunakan struktur IF-THEN-ELSE). Tampilkan jam J yang baru.

PENYELESAIAN

Penyelesaian masalah ini menggunakan prinsip konversi jam *(hh:mm:ss)* ke total detik. Tambahkan total detik dengan p, lalu konversi kembali total detik ke jam *(hh:mm:ss)*.

Algoritma JAM_TAMBAH_P

{ Mengisi nilai jam ke sebuah peubah, menambah dengan p menit }

DEKLARASI

```
Type jam : record <hh    : integer;      {0 . . 23}
                  mm   : integer;      {0 . . 59}
                  ss   : integer;      {0 . . 59}
                >
J : Jam
P , total_detik : integer
```

DESKRIPSI

Read(J. hh,J.mm,J.ss)
Read(p)
Total_detik ← (J. hh*3600) + (J.mm*60) + J. ss + (p*60)

{ tentukan jam yang baru }
J.hh ← total_detik div 3600
Sisa ← total_detik mod 3600
J.mm ← sisa div 60
J.ss sisa ← mod 60
Write (J.hh, J.mm, J.ss)

STUDI KASUS 2

Tulislah algoritma untuk menuliskan teks lagu *Anak Ayam Turun N* dengan *N* dibaca dari papan Kunci. Setiap baris sair lagu dicetak di dalam struktur pengulangan.

Contoh: *N* = 10, maka sair lagu *Anak Ayam Turun 10* tercetak seperti di bawah ini (perhatikan baris terakhir sedikit berbeda dengan baris sair sebelumnya):

Anak Ayam Turun 10
Anak ayam turun 10, mati satu tinggal 9
Anak ayam turun 9, mati satu tinggal 8
Anak ayam turun 8, mati satu tinggal 7
Anak ayam turun 7, mati satu tinggal 6
Anak ayam turun 6, mati satu tinggal 5
Anak ayam turun 5, mati satu tinggal 4
Anak ayam turun 4, mati satu tinggal 3
Anak ayam turun 3, mati satu tinggal 2
Anak ayam turun 2, mati satu tinggal 1
Anak ayam turun 1, mati satu tinggal induknya

PENYELESAIAN

Masalah ini adalah mencetak *string* "Anak ayam turun k, mati satu tinggal k-1" di dalam badan pengulangan. Pada awalnya, k = N. pada setiap pengulangan, nilai k selalu dikurangi satu.
Ketika k = 1, pencetakan *string* ditangani secara khusus, karena baris terakhir dari lagu tersebut berbeda dengan baris sebelumnya.

Algoritma LAGU_ANAK_AYAM
{ Mencetak lagu "Anak Ayam Turun N" }

DEKLARASI

```
N : integer

DESKRIPSI:
  Read (N)
  k ← N
  Write ( ' Anak Ayam Turun   ', N)
  While k > 1 do
     Write ( 'Anak ayam turun ', k, ', mati satu tinggal  ', k-1 )
     k ← k – 1
  Endwhile
  { k = 1 }

  if   k = 1  then
     write ( 'Anak ayam turun ', k, ', mati satu tinggal induknya, ' )
  endif
```

STUDI KASUS 3

Seorang pengirim surat menuliskan nama kota pada amplop surat tetapi tidak mencantumkan kode pos-nya.
Buatlah algoritma yang menerima masukan nama kota dan menuliskan kode pos kota tersebut ke piranti keluaran. Kota-kota yang tersedia di dalam table adalah sebagai berikut:

Padang	:	25000
Bandung	:	40100
Solo	:	51000
Denpasar	:	72000
Palu	:	92300

PENYELESAIAN

```
Algoritma KODE_POS
{ Menerima masukan nama kota dan mencetak kode pos ybs }

DEKLARASI
  Kota : string

DESKRIPSI:
  read   (kota)
  case   (kota)
    Kota  =  'padang'   : write ( '25000' )
    Kota  =  'bandung'  : write ( '40100' )
    Kota  =  'solo'     : write ( '51000' )
    Kota  =  'denpasar' : write ( '72000' )
    Kota  =  'palu'     : write ( '92300' )
  Endcase
```

STUDI KASUS 4

Di SMA (sekarang SMU) anda tentu masih ingat akar-akar persamaan kuadrat.

$ax^2 + bx + c = 0$

dapat dihitung dengan rumus *abc*, yaitu

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \; ; \; x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

Jenis akar bergantung pada nilai b^2 - 4ac (yang disebut diskriminan atau D):

- jika $b^2 - 4ac < 0$, maka akar-akarnya imaginer
- jika $b^2 - 4ac > 0$, maka akar-akarnya riil dan berbeda, $x_1 \neq x_2$
- jika $b^2 - 4ac = 0$, maka akar-akarnya riil dan kembar, $x_1 = x_2$

Misalnya persamaan kuadrat $x^2 - 4x - 5 = 0$:

$a = 1, b = -4, c = -5$

Nilai determinannya :

$D = b^2 - 4ac = (-4)^2 - 4(1)(-5) = 16 + 20 = 36$

Ternyata $b^2 - 4ac > 0$. jadi akar-akarnya riil dan berbeda, yaitu :

$$x_1 = \frac{-(-4) + \sqrt{36}}{2(1)} = 5 \quad \text{dan} \quad x_2 = \frac{-(-4) - \sqrt{36}}{2(1)} = -1$$

Tulislah algoritma untuk mencari akar persamaan kuadrat. Sebagai data masukannya adalah koefisien persamaan, *a, b,* dan *c*, sedangkan keluarannya adalah akar-akar persamaannya.

PENYELESAIAN

Analisis kasus terhadap permasalahan ini menghasilkan tiga macam kasus:

1. Kasus $D < 0$, akar-akarnya imaginer
2. Kasus $D > 0$, akarnya riil dan berbeda
3. Kasus $D = 0$, akar-akarnya kembar (sama)

Algoritma AKAR_PERSAMAAN_KUADRAT

{ Menghitung akar-akar persamaan kuadrat ax2 + bx + c = 0. Nilai a, b, dan c, dibaca dari piranti masukan, sedangkan akar-akarnya ditulis ke piranti keluaran }

DEKLARASI

```
a, b, c,   : real    { koefiesien persamaan kuadrat }
D          : real    { determinan }
x1, x2     : real    { akar-akar persamaan kuadrat }
```

DESKRIPSI

```
read (a, b, c)          { baca koefiesien persamaan kuadrat }
D ← b*b – 4*a*c     { hitung determinan }
if  D < 0  then
    Write ( 'Akar-akar persamaan kuadrat imaginer!' )
else
      if  D > 0 then
        { dua akar riil berbeda }
          x1 ← ( - b + SQRT (D) ) / (2*a)
          x2 ← ( - b – SQRT (D) ) / (2*a)
      else { D = 0 }
          { dua akar riil kembar }
            x1 ← ( - b + SQRT (D) ) / (2*a)
            x2 ← x1
      endif
      write (x1,x2) { cetak akar ke piranti keluaran }
endif
```

STUDI KASUS 5

Tulislah algoritma untuk menghitung jumlah deret

$$1 – 1/3 + 1/5 – 1/7 + 1/9 + \ldots \pm 1/N$$

Harga *N* adalah bilangan bulat positif yang dibaca dari piranti masukan. Jumlah deret dicetak ke piranti keluaran.

PENYELESAIAN

Perhatikan keteraturan deret tersebut: penyebut pada tiap suku deret bertambah 2 dari penyebut suku sebelumnya. Tanda suku berselang-seling, positif, negatif, positif, negatif dan seterusnya. Tanda positif adalah suku ke-*k* ganjil (k = 1,3, 5, …) dan tanda negatif untuk suku ke-*k* genap (k = 2, 4, 6, …). Bilangan ganjil dan genap dapat ditentukan dengan operator mod (ingat kembali soal menentukan bilangan ganjil dan genap pada Bab 5). Karena setiap suku berbentuk 1/x, maka setiap bilangan penyebut suku ke-*k* dapat ditulis dalam rumus umum:

suku ke-*k* = 1/k ; x = 1,3, . . ., N)

Pengulangan dilakukan selama x ≤ N.

Algoritma JUMLAH_DERET

{ Menghitung jumlah deret yang tanda sukunya berselang-seling positif dan negatif :

$$1 + 1/3 – 1/5 + 1/7 – 1/9 + \ldots \pm 1/N$$

Nilai N dibaca dari piranti masukan. Jumlah deret dicetak ke piranti keluaran }

DEKLARASI

```
sigma : real      { jumlah deret }
N     : integer   { penyebut suku terakhir, N > 0 }
k     : integer   { pencatat suku deret }
x     : integer   { bilangan penyebut suku ke-k }
```

DESKRIPSI:

```
read (N)
sigma ← 0
k ← 1
x ← 1
While x ≤ N do
    if k mod 2 = 1 then        { suku ke-k ganjil }
      sigma ← sigma + 1/x
    else                       { suku ke-k genap }
      sigma ← sigma – 1/x
    endif
    k ← k + 1                  { suku berikutnya }
    x ← x + 2                  { bilangan penyebut suku berikutnya }
endwhile                       { x > N }
write (sigma)
```

Algoritma di atas dapat disederhanakan lagi dengan mengingat bahwa tanda suku berselang-seling (analog dengan *on/off/on/off* . . .). didefinisikan sebuah nama peubah bernama tanda. Untuk suku pertama, nilainya:

```
tanda ← +1      { positif }
```

maka suku-suku berikutnya bertanda

```
tanda ← - tanda   { -1 x harga tanda suku sebelumnya }
```

Dengan demikian, pemeriksaan suku ke-k ganjil atau genap dengan notasi if-then dapat dihilangkan.

Algoritma JUMLAH_DERET

{ Menghitung jumlah deret yang tanda sukunya berselang-seling positif dan negatif :

$1 + 1/3 – 1/5 + 1/7 – 1/9 + . . . \pm 1/N$

Nilai N dibaca dari piranti masukan. Jumlah deret dicetak ke piranti keluaran }

DEKLARASI

```
sigma : real      { jumlah deret }
```

```
N      : integer      { penyebut suku terakhir }
k      : integer      { bilangan penyebut suku deret }
tanda  : integer      { tanda suku, positif atau negatif }
```

DESKRIPSI:

```
read (N)
sigma ← 0            { inisialisasi sigma }
tanda ← +1           { tanda suku pertama }
x     ← 1

while  x ≤ N  do
    sigma  ← sigma + tanda * 1/x
    x ← x + 2
    tanda← - tanda   { tanda suku berikutnya }
endwhile             { x > N }

write (sigma)
```

STUDI KASUS 6

Dibaca *N* buah bilangan bulat sembarang dari piranti masukan.
Tulislah algortima untuk menghitung jumlah nilai bilangan yang genap-genap saja.

Contohnya, jika bilangan yang kita baca adalah *(N = 7)*

5 10 47 2 8 20 23

maka total nilai bilangan yang genap-genap saja adalah

10 + 2 + 8 + 20 = 40

PENYELESAIAN

Algoritma untuk menentukan bilangan genap sudah pernah kita kemukakan di dalam Bab Analisis khusus. Misalkan bilangan bulat yang dibaca dari piranti masukan disimpan di dalam nama peubah bil, maka bilangan bulat yang genap ialah jika

Bil mod 2 = 0

Tiap kali bil dibaca dari piranti masukan, ia langsung dijumlahkan dengan jumlah nilai bilangan sebelumnya. Bila jumlah nilai bilangan bulat disimpan dalam peubah jumlah_nilai, maka total nilai bilangan bulat sekarang dinyatakan aksi

Jumlah_nilai ← total_nilai + bil

Algoritma JUMLAH BILANGAN GENAP

{ Menghitung jumlah nilai bilangan bulat yang dibaca dari piranti masukan. Banyaknya bilangan yang dibaca adalah N. Jumlah nilai bilangan genap dicetak ke piranti keluaran }

DEKLARASI

```
N           : integer   { banyak bilangan yang dibaca }
Bil         : integer   { bilangan bulat yang dibaca }
Jumlah_nilai : integer  { jumlah nilai bilangan genap }
i           : integer   { mencatat jumlah pembacaan data }
```

DESKRIPSI

```
    read (N)
    Jumlah_nilai ← 0
    i ← 1                   { mulai dengan pembacaan data pertama }

while  i ≤ N  do
    read ( bil )            { baca sembarang bilangan bulat }

    { jumlahkan bil yang genap saja }
    if bil mod 2 = 0 then   { bil adalah bilangan genap }
        jumlah_nilai ← jumlah_nilai + bil
    endif
    i ← i + 1       { naikkan pencatat pembacaan data berikutnya }
    endwhile        { i > N }

    write (jumlah_nilai)
```

STUDI KASUS 7

Pada umumnya algoritma menerima masukan dari pengguna. Masukan dari pengguna belum tentu sesuai dengan batasan nilai yang dibolehkan. Misalkan sebuah algoritma menerima masukan nilai bilangan bulat. Masukan yang dapat diterima hanya dari 10 sampai 20, di luar itu masukan ditolak. Bila masukan ditolak, pengguna harus memasukkan kembali nilai sampai benar. Algoritma pemeriksaan nilai masukan ini disebut ***algoritma validasi masukan.***

PENYELESAIAN

Algoritma VALIDASI_DATA_MASUKAN

{ Memvalidasi masukan yang diberikan oleh pengguna }

DEKLARASI

```
    Nilai   : integer
```

DESKRIPSI:

```
repeat
write ('masukan (10 – 20) ? ')
read (nilai)
if  (nilai < 10) or (nilai > 20) then
  write ('Masukan di luar rentang. Ulang lagi!')
```

```
endif
  Until (nilai >=10) and (nilai <=20)
  {Masukan sudah benar, silakan digunakan untuk proses selanjutnya }
```

STUDI KASUS 8

Di jurusan tertentu di sebuah Universitas ada N orang mahasiswa. Tiap mahasiswa mengambil m buah mata kuliah yang sama. Tulislah algoritma untuk menentukan nilai rata-rata tiap-tiap mahasiswa untuk seluruh mata kuliah.

Contoh: N = 3, M = 4

Nama	M-K 1	M-K 2	M-K 3	M-K 4	Rata-rata
Ahmad	40.0	80.0	20.0	60.0	50.0
Santi	80.0	45.0	90.0	75.0	65.0
Kuncoro	55.0	45.0	65.0	35.0	55.0

Keterangan:
M-K = Mata Kuliah

PENYELESAIAN

Algoritma RATA_RATA_NILAI_TIAP_MAHASISWA
{ Menentukan rata-rata nilai ujian tiap mahasiswa. }

DEKLARASI

```
N      : integer   { jumlah mahasiswa }
M      : integer   { jumlah mata kuliah }
nama   : string    { nama mahasiswa }
nilai  : real      { nilai ujian }
jumlah : real      { jumlah nilai ujian }
rerata : real      { rata-rata nilai ujian }
j, k   : integer   { pencacah pengulangan }
```

DESKRIPSI:

```
read (N)
Read (M)
j ← 1
While j ≤ N do
    read (nama)
    k ← 1  { mulai dari mata kuliah pertama }
    jumlah ← 0;
    while k ≤ M do
        read (nilai)
        jumlah ← jumlah + nilai
        k ← k+1   { ulangi untuk mata kuliah berikutnya )
    endwhile      { k > m }

    rerata ← jumlah/M
```

```
        write (rerata)

        j ← j+1          { ulangi untuk mahasiswa berikutnya }
    endwhile             { j > N }
```

STUDI KASUS 9

Tulislah algoritma untuk menghitung perpangkatan

a^n = a x a x a x a x . . . x a *{ sebanyak n kali }*

Nilai a dan n (n>0) dibaca dari piranti masukan.
Contohnya.

5^4 = 5 x 5 x 5 x 5 = 625
2^8 = 2 x 2 x 2 x 2 x 2 x 2 x 2 x 2 = 256

PENYELESAIAN

Studi kasus ini hampir mirip dengan menghitung jumlah deret 1 + 2 + … + N (lihat Bab 7 Pengulangan). Bedanya, di sini tiap bilangan dikalikan dan bilangan yang dikalikan selalu sama *(= a)*. Nilai perkalian sekarang adalah nilai perkalian sebelumnya dikali *a*.

Misalkan nilai perkalian disimpan di dalam peubah bernama pangkat. Maka nilai perkalian berikutnya adalah

Pangkat ← pangkat * 2

Peubah pangkat ini diinisialisasi dengan 1 (bukan 0). Aksi perkalian ini kita ulang sebanyak n kali.

Algoritma PERPANGKATAN
{ Menghitung perpangkatan an, dengan n > 0. Nilai a dan n dibaca dari piranti masukan. Nilai perpangkatan dicetak ke piranti keluaran. }

DEKLARASI

```
a       : real      { bilangan yang dipangkatkan }
n       : integer   { pemangkat, n > 0 }
pangkat : real      { hasil perpangkatan }
i       : integer   { pencatat pengulangan }
```

DESKRIPSI

```
read (a)
read (n)
pangkat  1
for  i ← 1 to n do
         Pangkat  pangkat * a
endfor
write (pangkat)
```

STUDI KASUS 10

Faktoral sebuah bilangan bulat tidak negatif n didefinisikan sebagai

$n ! = 1 \times 2 \times 3 \times 4 \times \ldots \times n \quad , n > 0$

dan khusus untuk n = 0! Didefinisikan sebagai

$0 ! = 1$

Tulislah algoritma untuk menghitung faktorial n. Harga n dibaca dari piranti masukan.

PENYELESAIAN

Untuk n = 0, maka faktorial (0) langsung diperoleh = 1. Untuk n>0, algortimanya mirip dengan Studi Kasus 9. perbedaannya, tiap bilangan yang dikalikan nilainya bertambah 1 dari bilangan sebelumnya. Misalkan nilai faktorial disimpan di dalam peubah fak, jika bilangan sekarang adalah k, maka nilai faktorial sekarang adalah :

fak ← fak * k

Dengan fak di inisialisasi dengan 1.

PENYELESAIAN

Algoritma FAKTORIAL

{ Menghitung n! untuk n bilangan bulat tidak negatif. Nilai n dibaca dari piranti masukan. Nilai faktorial dicetak ke piranti keluaran }

DEKLARASI

```
n    :  integer      { n nilai negatif }
fak  :  integer      { nilai faktorial bilangan n }
k    :  integer      { pencatat pengulangan }
```

DESKRIPSI:

```
read (n)
if  n = 0  then          { kasus khusus, n = 0 }
  fak ← 1                { 0! = 1 }
else  { n > 0 }
      fak ← 1
      for  k ← 1 to n do
           fak ← fak * k
      endfor
endif
write (fak)
```

STUDI KASUS 11

Dibaca data usia N orang siswa SMU dari piranti masukan. Tulislah algoritma untuk menghitung jumlah siswa yang berusia 15 tahun, jumlah siswa yang berusia 16 tahun, jumlah siswa yang berusia 17 tahun, jumlah siswa yang berusia 18 tahun, dan jumlah anak yang berusia selain itu.

PENYELESAIAN

Jumlah siswa yang berusia 15, 16, 17, 18, dan selain itu kita simpan dalam lima nama peubah yang berbeda-beda, misalnya

usia15, usia16, usia17, usia18, lainnya

Kelima nama peubah tersebut dapat kita analogikan dengan lima buah kotak. Keadaan awal, kelima kotak itu masih kosong. Keadaan ini sama dengan menginisialisasi kelima peubah / variabel dengan 0:

```
usia15   ← 0
usia16   ← 0
usia17   ← 0
usia18   ← 0
lainnya  ← 0
```

Setiap kali usia siswa dibaca dari piranti masukan, usia itu diperiksa apakah 15, 16, 17, 18, atau lainnya. Bila usia yang dibaca adalah 16, peubah usia 16 ditambah nilainya dengan 1:

```
usia16 ← usia16 + 1
```

Instruksi ini dianalogikan dengan memasukkan sebutir batu ke dalam kotak yang bernama usia16. Setiap kali usia yang dibaca berharga 16, masukkan sebutir batu ke dalam kotak. Jumlah batu di dalam kotak sekarang bertambah satu dari jumlah batu sebelumnya. Hal yang sama juga berlaku untuk kotak yang lain. Setelah N buah pembacaan selesai dilakukan, jumlah batu di dalam masing-masing kotak menyatakan jumlah siswa yang mempunyai usia tertentu.

Algortima STATISTIK_USIA

{ Menghitung jumlah siswa yang masing-masing berusia 15, 16, 17, 18, dan selain itu. Data usia siswa dibaca dari piranti masukan. Banyaknya siswa adalah N }

DEKLARASI

```
n        : integer                 { banyak data }
usia     : integer                 { usia siswa (tahun) }
usia15,usia16, usia17, usia18, lainnya : integer
i        : integer                 { pencatat pembacaan }
```

DEKLARASI :

```
read (N)
```

```
usia15   ← 0
usia16   ← 0
usia17   ← 0
usia18   ← 0
lainnya  ← 0
i ← 1

while i ≤ N do
    read (umur)
    case (umur)
        usia = 15  : usia15 ← usia15 + 1
        usia = 16  : usia16 ← usia16 + 1
        usia = 17  : usia17 ← usia17 + 1
        usia = 18  : usia18 ← usia18 + 1
        otherwise lainnya   ← lainnya + 1
    endcase
    i ← i + 1
endwhile          ( i > N )

write ( usia15, usia16, usia17, usia18, lainnya )
```

STUDI KASUS 12

Dibaca sekumpulan bilangan bulat positif dari piranti masukan. Tulislah algoritma untuk menentukan bilangan yang terbesar (maksimum). Banyak data bilangan tidak diketahui, tetapi proses pembacaan dan penentuan bilangan terkecil berakhir jika bilangan yang dibaca dari papan kunci adalah 0.

PENYELESAIAN

Asumsikan nilai maksimum sementara (maks) adalah bilangan bulat negatif terkecil. Bacalah data bilangan (x) dari piranti masukan. Setiap kali pemasukan data, bandingkan data tersebut dengan maks untuk menentukan data bilangan terbesar. Pada akhir pengulangan, maks berisi data terbesar.

Algoritma MAKSIMUM

{Menentukan data bilangan bulat positif terbesar dari sekumpulan data yang dibaca dari piranti masukan. Akhir pemasukan data adalah 0 }

DEKLARASI:

```
x     : integer      { bilangan bulat yang dibaca dari piranti masukan }
maks  : integer
```

DESKRIPSI:

```
maks ← 9999  { asumsi nilai terbesar sementara }
read(x)
while x ≠ 0 do { baca x selama bukan 0 }
        if x > maks then
```

```
                maks ← x
        endif
                read(x)
endwhile        { x = 0 }

write (maks)
```

STUDI KASUS 13

Tulislah algoritma untuk menghitung selisih waktu selesai bicara dengan waktu mulai bicara (hh:mm:ss) dari sebuah pembicaraan di sebuah warung telekomunikasi (wartel). Misalnya

Waktu 1	Waktu 2	Selisih (Waktu 2 – Waktu 1)
08:40:12	08:45:36	00:05:24
08:40:54	08:42:10	00:01:14
08:40:40	10:20:36	01:39:56

Asumsi:
hh.Waktu2 > hh.Waktu1.

Algoritma perhitungan selisih waktu menggunakan analisis kasus.

PENYELESAIAN
Algoritma menghitung selisih waktu dengan cara analisis kasus relatif lebih sulit dibandingkan dengan cara kalkulasi (lihat Bab URUTAN) terutama pada kasus detik2 < detik1, atau menit2 < menit1. Bila bertemu kasus seperti ini, kita "*terpaksa*" meminjam dari besaran waktu yang lebih tinggi.

Misalnya pada kasus detik2<detik1 seperti berikut:
(8:42:10) – (8:40:54)

Karena 10<54, maka pinjam satu menit dari 42. Satu menit itu sama dengan 60 detik, sehingga pengurangan detik menghasilkan
(10+60) – (54) = 16

Karena 42 menit sudah dipinjam satu menit, maka sisanya 41 menit, sehingga pengurangan menit menghasilkan
(10+60) – (54) = 16

Cara yang sama juga berlaku pada pengurangan (menit2 – menit1). Jika menit2 < menit1, pinjam satu jam (=60) menit dari waktu jam2. Cobalah anda lakukan pada pengurangan (10:20:36).

Algoritma SELISIH_WAKTU
{Menghitung selisih dua buah waktu: (j2:m2:d2) – (j1:m1:d1) data waktu dibaca dari piranti masukan}

DEKLARASI

```
type JAM : record < hh  : integer,  {0..23}
                    mm  : integer,  {0..59}
                    ss  : integer , {0..59}
                  >
W1 : JAM   { jam awal percakapan }
W2 : JAM   { jam selesai percakapan }
W3 : JAM   { lama percakapan }
```

DESKRIPSI:

```
{ baca jam awal dan jam selesai percakapan }
read(W1.hh, W1.mm, W1.ss)
read(W2.hh, W2.mm, W2.ss)

{ pengurangan detik }
if  W2.ss ≥ W1.ss then          { tidak ada masalah pengurangan detik }
    W3.ss ← W2.ss – W1.ss  { selisih detik }
else  { W2.ss < W1.ss }
    W3.ss ← (W2.ss + 60 ) – W1.ss     { pinjam satu menit dari menit 2,
                                        Lalu kurangi dengan detik 1 }
    W2.mm ← W2.mm – 1               { menit 2 berkurang satu karena dipinjam }
endif

{ pengurangan menit }
if  W2.mm ≥ W1.mm then                { tidak ada masalah pengurangan menit }
    W3.mm ← W2.mm – W1.mm         { selisih menit }
else  { W2.mm <  W1.mm }
    W3.mm ← (W2.mm + 60) – W1.mm   { pinjam satu jam dari jam 2,
                                     Lalu kurangi dengan menit 1 }
    W2.hh ← W2.hh – 1                 { jam 2 berkurang satu karena dipinjam }
endif

{ pengurangan jam }
W3.hh ← W2.hh – W1.hh

write(W3.hh,W3.mm,W3.ss)
```

STUDI KASUS 14

Lanjutkan algoritma pada Studi Kasus 11 sehingga dapat menghitung biaya percakapan yang tarif tiap pulsanya bergantung pada kode wilayah tujuan pembicaraan. Misalnya:

Kode	Wilayah Kota	Tarif Tiap Pulsa	Lama Pulsa
021	Jakarta	Rp 150	1 menit
0751	Padang	Rp 250	30 detik
0737	Medan	Rp 375	25 detik
0912	Balikpapan	Rp 415	20 detik
0981	Ternate	Rp 510	17 detik

Data masukan tambahan adalah kode wilayah.

PENYELESAIAN
Perhitungan pulsa dan biaya pembicaraan adalah
Pulsa = lama pembicaraan /lama pulsa (bergantung wilayah)
Biaya = pulsa * tarif pulsa (bergantung wilayah)

Algorit ma HITUNG_BIAYA_PERCAKAPAN
{Menghitung biaya percakapan di warung telekomunikasi (wartel). Tarif tiap pulsa bergantung pada wilayah tujuan percakapan. Biaya percakapan dihitung dari lama percakapan }

DEKLARASI

```
type JAM : record < hh : integer,   {0..23}
                    mm : integer,   {0..59}
                    ss : integer    {0..59}
                  >
W1 : JAM   { jam awal percakapan }
W2 : JAM   { jam selesai percakapan }
W3 : JAM   { lama percakapan }

kode_wil : string   { kode wilayah tujuan percakapan }
lama     : integer  { lama pembicaraan dalam detik }
pulswil  : real     { ukuran pulsa, bergantung kode wilayah }
tarifwil : real     { tarif per pulsa, bergantung pada kode wilayah }
biaya    : real     { biaya percakapan }
```

DESKRIPSI:

```
read (W1.hh, W1.mm, W1.ss)      { jam awal percakapan }
read (W2.hh, W2.mm, W2.ss)      { jam selesai percakapan }
read (kode_wil)                 { kode wilayah tujuan percakapan }

{ pengurangan detik }
if  W2.ss ≥ W1.ss then          { tidak ada masalah pengurangan detik }
    W3.ss ← W2.ss – W1.ss       { selisih detik }
else   {W2.ss < W1.ss}
    W3.ss ← (W2.ss + 60) – W1.ss   { pinjam aatu menit dari menit 2,
                                     Lalu kurangi dengan detik 1 }
    W2.mm ← W2.mm – 1              { menit 2 berkurang satu karena dipinjam }
endif

{ Pengurangan menit }
if  W2.mm ≥ W1.mm  then         { tidak ada masalah pengurangan menit }
    W3.mm ← W2.mm – W1.mm       { selisih menit }
else   { W2.mm < W1.mm }
    W3.mm ← (W2.mm + 60) – W1 . mm   { pinjam satu jam dari jam 2,
                                       Lalu kurangi dengan menit 1 }
    W2.hh ← W2.hh – 1           { jam 2 berkurang satu karena dipinjam }
endif
```

```
{ pengurangan jam }
W3.hh ← W2.hh – W1.hh

{ hitung lama percakapan dalam total detik }
lama ← (W3.hh*3600) + (W3.mm*60) + W3.ss

{tentukan ukuran pulsa dan tarif tiap pulsa, bergantung pada kode wilayah tujuan percakapan }

case (kode_wil)
     kode_wil='021'   : pulswil ← 60
                        tarifwil ← 150
     kode_wil='0751'  : pulswil ← 30
                        tarifwil ← 250
     kode_wil='0737'  : pulswil ← 25
                        tarifwil ← 375
     kode_wil='0912'  : pulswil ← 20
                        tarifwil ← 415
     kode_wil='0981'  : pulswil ← 17
                        tarifwil ← 510
endcase

{ hitung jumlah pulsa dan biaya untuk seluruh pulsa }
pulsa  ← lama/pulswil
biaya  ← pulsa*tarifwil

write(W3.hh, W3.mm, W3.ss, biaya)
```

STUDI KASUS 15

Bulan Februari mempunyai jumlah hari yang unik. Jumlah harinya ada yang 28 hari dan ada yang 29 hari. Bulan Februari mempunyai jumlah hari 29 bila berada pada tahun kabisat (tahun yang habis dibagi empat). Pada tahun yang bukan tahun kabisat, jumlah harinya 28.
Dibaca sebuah penanggalan pada bulan Februari, tentukan tanggal pada hari berikutnya.

Contoh:

Sekarang	17-2-1999
Besok	18-2-1999
Sekarang	28-2-1999
Besok	1-3-1999
Sekarang	28-2-1996
Besok	29-2-1996 (awas, tahun kabisat!)

PENYELESAIAN

Algoritma TANGGAL_BESOK_BULAN_FEBRUARI
{ Menentukan tanggal keesokan hari dari tanggal sekarang pada bulan Februari }

DEKLARASI

```
type tanggal : record <  dd  : integer, {1..31}
                         mm  : integer, {1..12}
                         yy  : integer, { > 0 }
                      >

T : tanggal
```

DESKRIPSI:

```
T.mm  ← 2        { bulan Februari }
read(T.dd.T.yy)
write('Tanggal sekarang: ', T.dd,'-',T.mm,'-',T.yy)

case (T.dd)
    T.dd < 28 : T.dd ← T.dd + 1
    T.dd = 28 : {bergantung tahunnya}
    case (T.yy)
        T.yy mod 4 = 0 :{kabisat}
            T.dd  ← T.dd + 1
        T.yy mod 4 ≠ 0 : {bukan kabisat}
            T.dd  ← 1
            T.mm  ← T.mm + 1
    endcase
    T.dd = 29 : T.dd  ← 1              {tanggal 1 bulan Maret}
                T.mm  ← T.mm + 1       { bulan Maret }
endcase

write('Tanggal besok: ', T.dd,'-',T.mm,'-',T.yy)
```

JIKA MENGGUNAKAN STRUKTUR IF-THEN-ELSE

DESKRIPSI:

```
T.mm ← 2        { bulan Februari }
read(T.dd,T.yy)
write('Tanggal sekarang: ', T.dd,'-', T.mm,'-',T.yy)

if T.dd < 28 then       { tidak ada masalah dengan tahun }
    T.dd ← T.dd + 1     { tanggal besok }
else
    if  T.dd = 28 then          {tanggal besok bergantung pada tahun kabisat}
        if  T.yy mod 4 = 0 then {tahun kabisat, maka besok 29}
            T.dd ← T.dd + 1     {29}
        else                    { bukan tahun kabisat, jadi sesudah 28 langsung 1 }
            T.dd ← 1            { tanggal 1 bulan Maret }
```

```
            T.mm ← T.mm + 1     { bulan Maret }
        endif
    else
        if  T.dd = 29 then
            T.dd ← 1            { tanggal 1 bulan Maret }
            T.mm ← T.mm + 1     { bulan Maret }
        endif
    endif
endif
```

STUDI KASUS 16

Masalah studi kasus 15 yang diperluas, sehingga dapat menghitung tanggal berikutnya dari tanggal sekarang untuk sembarang bulan. Misalnya,

Tanggal Sekarang	Tanggal Besok
13-6-1996	1-5-1986
1-2-1991	31-12-1996
1-1-1993	

Tanggal sekarang dibaca dari piranti masukan.

PENYELESAIAN

Mula-mula kita harus memilih bulan-bulan berdasarkan jumlah harinya karena tiap bulan tidak sama jumlah harinya:

Bulan	Jumlah hari
1, 3, 5, 7, 8, 10, 12	31 hari
4, 6, 9, 11	30 hari
2	28 atau 29 hari,bergantung tahun kabisat.

Pada tahun kabisat, jumlah hari dalam bulan Februari adalah 28, sedangkan pada tahun bukan kabisat jumlah hari dalam bulan Februari adalah 29.
Disamping itu kita juga harus menangani kasus pergantian tahun, misalnya pada contoh pergantian tanggal 31-12-1992 mennjadi 1-1-1993

Algoritma TANGGAL_BESOK
{Menentukan tanggal berikutnya setelah tanggal sekarang. Tanggal sekarang dibaca dari piranti masukan }

DEKLARASI

```
type tanggal : record < dd : integer, {1..31}
                       mm : integer, {1..12}
                       yy : integer  { >0 }
                     >
```

```
    T : tanggal

DESKRIPSI:
    read (T.dd, T.mm, T.yy)
    case (T.mm)
        T.mm = [4, 6,9,11]  :  { sebulan = 30 hari }
            case (T.dd)
                T.dd < 30 : T.dd ← T.dd + 1
                T.dd = 30 : T.dd ← 1
                T.mm  ← T.mm + 1
            endcase
        T.mm =  [1,3,5,7,8,10,12]:  { sebulan 31 hari }
            case (T.dd)
                T.dd < 31 : T.dd ← T.dd + 1
                T.dd = 31 : T.dd ← 1
                T.mm  ← T.mm  + 1
            endcase
        T.mm = 2  :  { bulan Februari }
            case (T.dd)
                T.dd < 28  :  T.dd ← T.dd + 1
                T.dd = 28  :  { bergantung tahunnya }
                case (T.yy)
                    T.yy mod  4 = 0 :  T.dd ← T.dd + 1        {kabisat}
                    T.yy mod  4 ≠ 0 :  {bukan kabisat}
                                       T.dd   ← 1
                                       T.mm  ← T.mm + 1
                Endcase
                T.dd  = 29 :  T.dd   ← 1                 { tanggal 1 bulan Maret }
                              T.mm  ← T.mm + 1       { bulan Maret }
            Endcase

        T.mm = 12:  { mungkin pergantian tahun }
            case (T.tgl)
                  T.dd  < 31 :  T.dd   ← T.dd + 1
                  T.dd  = 31  :  T.dd   ← 1
                                 T.mm  ← 1    {Januari}
                                 T.yy   ← T.yy + 1
            endcase
    endcase

    { cetak tanggal berikutnya itu }
    write (T.dd, '-', T.mm, '-', T.yy)
```

STUDI KASUS 17

Buatlah algoritma yang meniru mekanisme pembacaan kata sandi (*password*) dari piranti masukan. Kata sandi disimpan didalam nama tetapan. Apabila kata sandi yang dibaca salah, maka pembacaan kata sandi boleh diulang maksimum 3 kali

PENYELESAIAN

Misalkan pw adalah kata sandi yang dibaca dari piranti masukan. Definisikan sebuah peubah bertipe *boolean* yang bernama sah. Peubah sah berharga *false* jika pw belum sama dengan kata sandi, dan akan berharga *true* jika sebaliknya. Jika pw tidak sama dengan kata sandi, naikkan pencacah pembacaan pw. Proses pembacaan pw diulangi sampai sah berharga *true* atau pencacah pembacaan sudah melebihi tiga.

Algoritma KATA_SANDI
{Meniru mekanisme pembacaan kata sandi dari piranti masukan. Maksimum pembacaan kata sandi adalah tiga kali }

```
DEKLARASI
    const sandi  = 'abcdef'    { kata sandi }
    pw    : string      { kata sandi yang dibaca dari piranti masukan }
    sah   : boolean     { false jika pw belum sama dengan sandi, true jika sebaliknya }
    k     : integer     { pencatat jumlah pembacaan pw }

DESRKIPSI:
    k   ← 1
    sah ← false
    repeat
        read (pw)
        if pw = sandi then
            sah ← true
        else                        { pw ≠ sandi }
            write                   {'Kata sandi salah, ulangi lagi (maks 3 kali)'}
            k ← k + 1
        endif
    until (sah) or (k > 3)
```

Apa pendapat anda bila notasi pengulangan **repeat-until** diganti dengan **while-do** sebagai berikut :

(A)
```
k ← 1
sah ← false
while (not sah) and (k ≤ 3) do
    read (pw)
    if pw = sandi then
        sah ← true
    else { pw ≠ sandi }
```

```
            write ( 'kata sandi salah, ulangi lagi' )
            k ← k + 1
        endif
    endwhile
    { sah  or  k > 3 }
```

(B)
```
k ← 1
sah ← false
read (pw)
while (not sah) and (k ≤ 3) do
    if  pw = sandi then
        sah ← true
    else  { pw ≠ sandi }
        write ( 'kata sandi salah, ulangi lagi' )
        k ← k + 1
        read (pw)
    endif
endwhile
{ sah  or  k > 3 }
```

BAB IX
PROSEDUR

9.1 DEFINISI :

- Modul Program mengerjakan tugas/aktifitas yang spesifik dan menghasilkan suatu efek *netto*.
- Suatu efek netto diketahui dengan membandingkan keadaan awal dan akhir pada pelaksanaan sebuah prosedur.
- Oleh karena itu setiap prosedur harus didefinisikan keadaan awal sebelum rangkaian instruksi dilaksanakan dalam prosedur dan keadaan akhir yang diharapkan setelah rangkaian intruksi dilaksanakan

9.2 ILUSTRASI PROSEDUR

Untuk melakukan keberangkatan ke luar negeri, harus mengetahui prosedur dengan menggunakan pesawat terbang ?

Prosedur URUS PASSPORT (dikantor imigrasi)

- Isi formulir permintaan passport dengan lampiran foto copy KTP, KK, pas foto
- Serahkan Formulir yang sudah diisi beserta biaya pembuatan passport
- Wawancara dengan petugas imigrasi
- Terima Passport

Prosedur URUS VISA (dikantor kedutaan Besar)

- Isi formulir permohonan Visa, dilampiri foto copy KTP, passport, pas foto, tiket pesawat terbang
- Serahkan formulir yang diisi beserta biaya pengurusan Visa
- Terima Visa

Prosedur BERANGKAT DARI BANDARA

- Datang ke Bandara satu jam sebelum berangkat
- Jika akan naik pesawat, tunjukkan tiket, passport, visa ke petugas
- Naik ke Pesawat
- Sampai di negara tujuan

Jadi, bila keluar negeri, algoritmanya adalah sebagai berikut:

Algoritma PERGI_KE_LUAR_NEGERI

DESKRIPSI

- URUS PASSPORT
- URUS VISA
- BERANGKAT DARI BANDARA

MENDEFINISIKAN PROSEDUR

- Struktur Prosedur sama dengan Struktur Algoritma, yang terdiri dari
  - header (terdiri nama prosedur dan komentar)
  - Deklarasi
  - Badan Prosedur
- Setiap prosedur mempunyai nama yang unik
- Nama Prosedur sebaiknya diawali dengan kata kerja, misalnya HITUNG_LUAS, TUKAR, CARI_MAX dan lain-lain
- Notasi Algoritma untuk mendefinisikan struktur Prosedur

Procedure NAMA_PROSEDUR
{Spesifikasi prosedur, berisi penjelasan tentang yang akan dilakukan}
{Kondisi Awal : keadaan sebelum prosedur dilaksanakan}
{Kondisi Akhir : keadaan sesudah prosedur dilaksanakan}

DEKLARASI
{ semua nama yang dipakai dalam prosedur dan hanya berlaku LOKAL di dalam prosedur didefinisikan disini }

DESKRIPSI
{ badan prosedur, berisi kumpulan instruksi}

9.3 NAMA GLOBAL & LOKAL

Nama-nama (konstanta, variabel, type dll) yang dideklarasikan dalam bagian DEKLARASI Prosedur, prosedur hanya dikenal didalam *Body* prosedur. Nama-nama dibagian DEKLARASI prosedur bersifat LOKAL. (hanya dapat digunakan didalam prosedur yang melingkupi), sedang Nama-nama dibagian DEKLARASI program Utama bersifat GLOBAL. (dapat digunakan dimanapun dalam program, baik didalam program utama dan didalam prosedur)

9.3.1 Contoh Global & Lokal

Procedure HIT_RATA
{menghitung rata-rata N bil. Bulat yang dibaca dari keyboard }
{K.Awal : sembaramg }
{K. Akhir : rata-rata seluruh bil dicetak pada piranti keluaran}

DEKLARASI
x, k, jumlah : integer

DESKRIPSI

```
jumlah ← 0
k ← 1
while k ≤ N do
        read (x)
        jumlah ← jumlah + x
        k ← k + 1
endwhile
( k > N }
u ← jumlah / N
```

9.3.2 Contoh Rata-Rata Bilangan Bulat

Algoritma Rata_Bil_Bulat

{ Program Utama menghitung rata-rata N buah bil. Bulat }

```
DEKLARASI
  N : integer
  u  : real
  procedure HIT_RATA
  { menghitung rata-rata N bil. Bulat yang dibaca dari keyboard}

DESKRIPSI
  read (N)
  write ('menghitung nilai rata-rata'}
  HIT_RATA
  write ('nilai rata-rata = ', u)
```

- Pada prosedur dan program utama diatas, variabel N dan u di dalam DEKLARASI program utama. Karena itu N dan u bersifat GLOBAL, sehingga *dikenal* dan dapat digunakan dalam prosedur HIT_RATA
- Sebaliknya variabel x, k dan jumlah di DEKLARASI di dalam prosedur, sehingga ketiga variabel tersebut Bersifat LOKAL, dan hanya dikenal dan digunakan pada lingkup prosedur itu saja.
- Nama yang di DEKLARASI di dalam Prosedur dan Program Utama mungkin saja sama. Namun sifat LOKAL dan GLOBAL, tetap tidak berubah. Bila demikian nama dalam prosedur bersifat variabel LOKAL, dan diluar Prosedur berlaku sebagai variabel GLOBAL

9.4 PARAMETER

Parameter pada prosedur terdiri dari :

- Parameter **actual** : Parameter yang disertakan pada saat pemanggilan
- Parameter **formal** : Parameter yang di deklarasikan didalam prosedur

Saat pemanggilan prosedur, parameter aktual menggantikan parameter formal, setiap parameter aktual berpasangan dengan parameter formal yang berkesesuaian.
Prosedur dengan parameter diakses dengan cara memanggil nama prosedur dari program pemanggil (prog. Utama atau modul yang lain) disertai parameter aktualnya.

Contoh:

NAMA_PROSEDUR (daftar parameter aktual)

Aturan penting dalam korespondensi satu-satu antara parameter aktual dan parameter formal adalah sebagai berikut :

- Jumlah parameter aktual pada pemanggilan prosedur harus sama dengan jumlah parameter formal pada deklarasi prosedurnya.
- Tiap parameter aktual harus bertip sama dengan tipe parameter formal yang berkesesuaian
- Tiap parameter aktual harus diekspresikan dalam cara yang taat-asas dengan parameter formal yang berkesesuaian, bergantung pada jenis parameter formal

Berdasarkan penggunaannya, parameter formal yang disertakan dalam prosedur :

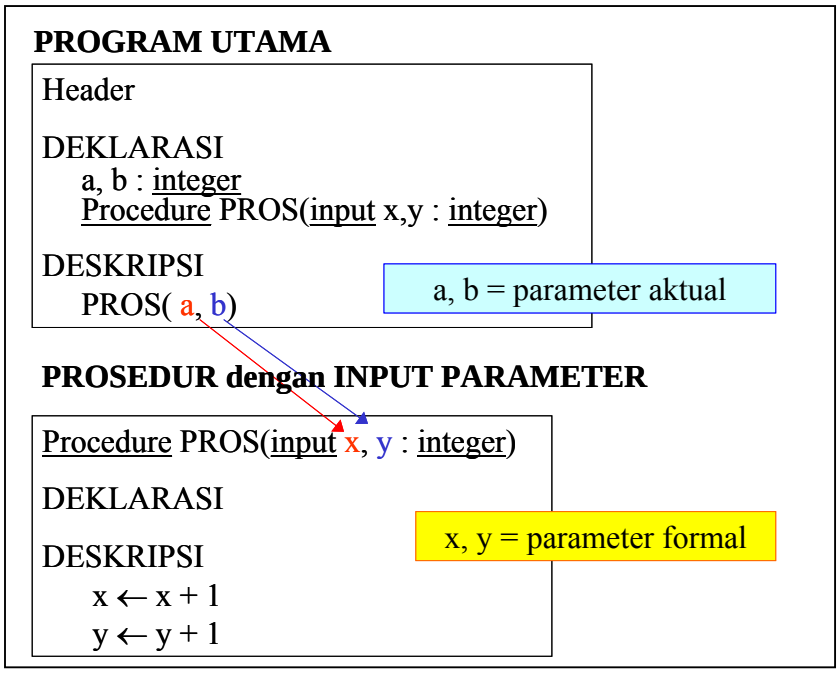
- Input Parameter : parameter yang nilainya sebagai input untuk prosedur, pada bahasa pemrograman disebut *parameter by value*
- Output Parameter : parameter yang menampung output yang dihasilkan oleh prosedur
- Input/Output Parameter : parameter yang berfungsi sbg input sekaligus sbg output dari prosedur tersebut Pada bahasa pemrograman disebut *parameter by reference*

9.4.1 INPUT PARAMETER

Nilai parameter aktual diisikan (*assign*) ke dalam parameter formal yang berkesesuaian. Nilai ini digunakan dalam prosedur yang bersangkutan. Nilai input parameter tak dapat dikirim dalam arah sebaliknya. Perubahan nilai parameter didalam prosedur tidak mengubah nilai parameter aktual. Karena yang dipentingkan adalah nilainya, maka nama parameter aktual boleh berbeda dng nama parameter formal yang berkesesuaian

CONTOH INPUT PARAMETER

```
Procedure SATU (input x,y: integer)
{prosedur dng parameter formal jenis input parameter}
{K. Awal : x & y sudah terdef}
{K. Akhir: x & y ditambah 1, dicetak pada piranti output}

DEKLARASI
{tak ada}

DESKRIPSI
        x ← x + 1
        y ← y + 1
        write(x)
        write(y)

{x dan y = parameter formal}

Algoritma PQR
{Program Utama yang memanggil prosedur SATU}

DEKLARASI
        a, b : integer
        Procedure SATU (input x,y:  integer)

DESKRIPSI
        SATU(4,10)          {SATU ke 1}
        read(a,b)
        SATU(a,b)           {SATU ke 2}
        SATU(a+5,17)        {SATU ke 3}
{4 dan 10 ; a dan b; a+5 dan 17 = parameter aktual }
```

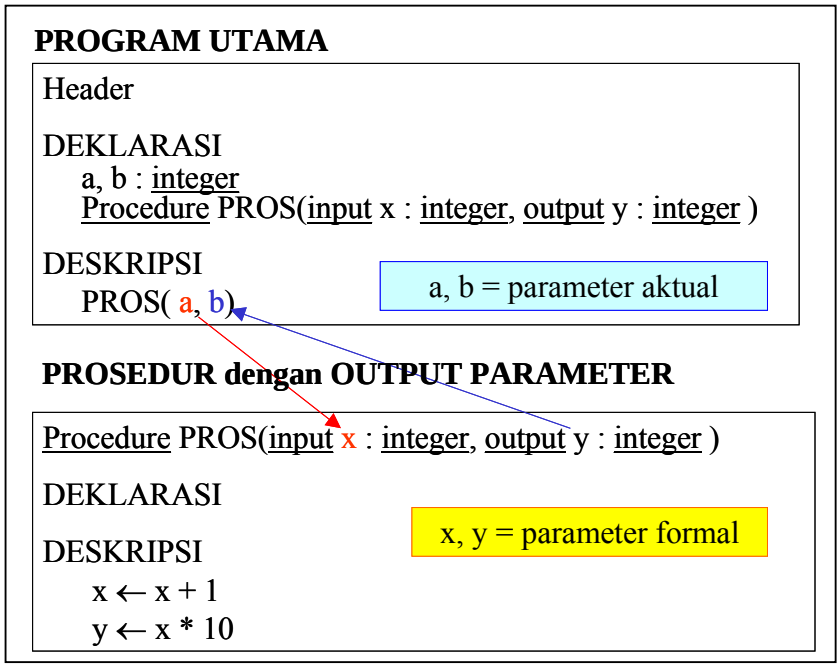


9.4.2 OUTPUT PARAMETER

Prosedur menghasilkan satu atau lebih nilai yang digunakan program pemanggil, nilai Output ditampung didalam Output Parameter. Bila prosedur yang mengandung Output Parameter dipanggil, parameter aktual didalam program pemanggil menggantikan parameter formal didalam prosedur, jadi parameter aktual akan digunakan dalam prosedur. Setelah pemanggilan, parameter aktual berisi nilai yang merupakan keluaran dari prosedur

CONTOH OUTPUT PARAMETER

```
Procedure DUA (input x: integer, output y : integer )
{prosedur dng parameter formal jenis output parameter}
{K. Awal : x & y sudah terdef}
{K. Akhir: x ditambah 1, hasil disimpan dalam y}

DEKLARASI
{tak ada}

DESKRIPSI
        x ← x + 1
        y ← x * 10

{x dan y = parameter formal}
Algoritma PQR
{Program Utama yang memanggil prosedur DUA}

DEKLARASI
        a, b : integer
        Procedure DUA (input x :                integer, output y: integer )

DESKRIPSI
        DUA(4, b)           {DUA ke 1}
        write(b)
        read(a)
        DUA(a,b)            {DUA ke 2}
        write(b)
        DUA(a+5,b)          {DUA ke 3}
        write(b)
```



9.4.3 INPUT/OUTPUT PARAMETER

Input parameter dikirim dari *program pemanggil* ke *prosedur* dan Output parameter hanya mengirim hasil dari *prosedur* ke *program pemanggil.* Sedang Input/Output parameter dikirim dari *program pemanggil* ke *prosedur* , juga mengirim hasil dari *prosedur* ke *program pemanggi.* Setelah pemanggilan, parameter aktual digantikan nilai dari parameter formal yang merupakan keluaran dari prosedur

CONTOH INPUT/OUTPUT PARAMETER

```
INPUT/OUTPUT PARAMETER (SATU)
Procedure TIGA (input x,y: integer)
{prosedur  jenis input parameter}
{K. Awal : x & y sudah terdef}
{K. Akhir: x ditambah 2, y dikurangi 2}

DEKLARASI
{tak ada}

DESKRIPSI
        x ← x + 2
        y ← y – 2
        write('nilai x dan y')
        write('x = ',x)
        write('y = ',y)
{x dan y = parameter formal}
Algoritma FGH
{Program Utama yang memanggil prosedur TIGA}
```

```
DEKLARASI
        a, b : integer
        Procedure TIGA (input x, y: integer )

DESKRIPSI
        a ← 15
        b ← 10
        write('a&b sebelum pros')
        write('a = ',a)
        write('b = ',b)
        TIGA(a,b)
        write('a&b sesudah pros')
        write('a = ',a)
        write('b = ',b)

INPUT/OUTPUT PARAMETER  (DUA)
Procedure TIGA (input/output x,y: integer)
{prosedur  jenis input/output parameter}
{K. Awal : x & y sudah terdef}
{K. Akhir: x ditambah 2, y dikurangi 2}

DEKLARASI
{tak ada}

DESKRIPSI
        x ← x + 2
        y ← y – 2
        write('nilai x dan y')
        write('x = ',x)
        write('y = ',y)
{x dan y = parameter formal}
Algoritma FGH
{Program Utama yang memanggil prosedur TIGA}

DEKLARASI
        a, b : integer
        Procedure TIGA (input/output x, y: integer )

DESKRIPSI
        a ← 15
        b ← 10
        write('a&b sebelum pros')
        write('a = ',a)
        write('b = ',b)
        TIGA(a,b)
        write('a&b sesudah pros')
        write('a = ',a)
        write('b = ',b)
```

INPUT/OUTPUT PARAMETER [Bandingkan dengan SATU dan DUA]
INPUT/OUTPUT PARAMETER (SATU)

- Nilai a dan b *sebelum* panggil prosedur
  - a = 15
  - b = 10
- Nilai x dan y pada TIGA
  - x = 17
  - y = 8
- Nilai a dan b *sesudah* panggil prosedur
  - a = 15
  - b = 10

- Nilai a dan b tidak berubah walaupun telah melaksanakan prosedur TIGA

INPUT/OUTPUT PARAMETER (DUA)

- Nilai a dan b *sebelum* panggil prosedur
  - a = 15
  - b = 10
- Nilai x dan y pada TIGA
  - x = 17
  - y = 8
- Nilai a dan b sesudah panggil prosedur
  - a = 17
  - b = 8

- Nilai a dan b diubah dalam prosedur TIGA, perubahan ini dibawa ke pemanggil

Penggunaan Prosedur

- Dianjurkan program menggunakan modular (prosedur/fungsi)
- Program dengan beberapa modul menunjukkan teknik pemrograman yang terstruktur
- Dianjurkan prosedur menggunakan parameter
- Bila prosedur menghasilkan output , gunakan output parameter
- Bila output hanya dalam prosedur, gunakan input parameter
- Bila prosedur menerima input dan output sekaligus pada parameter yang sama, gunakan input/output parameter

PROGRAM UTAMA

Header

DEKLARASI
a, b : integer
Procedure PROS(input / input x,y : integer)

DESKRIPSI
PROS(a, b)

a, b = parameter aktual

PROSEDUR dengan INPUT/OUTPUT PARAMETER

Procedure PROS(input / output x,y : integer)

DEKLARASI

DESKRIPSI
x ← x + 2
y ← y - 2

x, y = parameter formal

BAB X
FUNCTION

Merupakan Modul program yang mengembalikan (*return*) sebuah nilai yang bertipe sederhana (*integer, real, boolean dan string*)

- Definisi fungsi seperti fungsi matematika. Contoh:
  - $f(x) = 2x^2 + 5x - 8$
  - $H(x,y) = 3x - y + xy$
  - $x = 2 \rightarrow f(1) = 2.2^2 + 5.2 - 8 = 10$
  - $x = 1 ; y = 2 \rightarrow H(1,2) = 3.1 - 2 + 1.2 = 3$

Mendefinisikan Fungsi

Function Nama_Fungsi(input daftar para formal) → tipe hasil

{ spesifikasi fungsi, menjelaskan apa yang dikerjakan dan yang di return oleh fungsi }

DEKLARASI

{ semua nama yang dipakai didalam algoritma fungsi dideklarasikan di sini. Nama yang didefinisikan di dalam DEKLARASI Lokal hanya dikenal dan dipakai di dalam fungsi ini saja }

DESKRIPSI

{ badan fungsi, berisi instruksi2 untuk menghasilkan nilai yang akan dikembalikan oleh fungsi }

return hasil *{ mengembalikan nilai hasil dari fungsi }*

Memanggil FUNGSI

- Fungsi dipanggil dari program pemanggil, diikuti dengan daftar parameter aktual (*bila ada*).
- Nilai yang dihasilkan Fungsi, dapat disimpan dalam *variabel* yang bertipe = tipe Fungsi *var* ← *Nama_Fungsi*(daftar parameter aktual)
- Atau langsung digunakan seperti contoh :
  - write (Nama_Fungsi(daftar parameter aktual))
  - if Nama_Fungsi(daftar parameter aktual) < 0 then …
  - z ← 2 * Nama_Fungsi(daftar parameter aktual) + x

CONTOH FAKTORIAL

Fungsi untuk menghitung nilai faktorial dari bilangan bulat tidak negatif

$n! = 1 \quad , n = 0$

$= 1 \times 2 \times 3 \times \ldots\ldots \times (n\text{-}1) \times n \quad , n > 0$

Tulislah Fungsi untuk menghitung faktorial bilangan bulat n ($n \geq 0$)

```
function FAK(input n : integer) → integer
{mengembalikan harga n!, untuk n ≥ 0}

DEKLARASI
    k, p    :   integer

DESKRIPSI
    if  n = 0 then          {kasus khusus, n = 0 }
        return 1
    else
        p ← 1
        for k ← 1 to n do
            p ← p * k
        endfor
        return p
    endif
```

CONTOH PANGKAT (POWER)

Fungsi untuk menghitung perpangkatan x^n, $n \geq 0$, $x \in \mathbf{R}$

```
function POWER(input x : real , input n : integer) → integer
{mengembalikan harga perpangkatan x^n }

DEKLARASI
    p   : integer
    i   : integer

DESKRIPSI
    p ← 1
    for i ← 1 to n do
        p ← p * x
    endfor
    return p
```

CONTOH EXPONEN

Tulislah Fungsi untuk menghitung *exp(x)* yang didefinikan dengan deret berikut ini:

$$e^x \approx 1 + x! + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots + \frac{x^n}{n!}$$

ambil nilai $n = 10$ (semakin besar nilai n semakin teliti nilai *exp(x)*).
Nilai x dibaca dari piranti masukan, manfaatkan fungsi FAK dan POWER diatas

Tiap suku dinyatakan dalam bentuk :

$\frac{x^k}{k!}, k = 0,1,2,3,....,n$

Jadi, jumlah deret adalah $S \leftarrow S + \frac{x^k}{k!}$, yang dalam hal ini,

S diinisialisasi dengan 0
x^k = dihitung dengan fungsi POWER
k! = dihitung dengan fungsi FAK

CONTOH Fungsi EXPONEN

```
function EXP(input x : real) → real
{mengembalikan nilai exp(x) }

DEKLARASI
    const n : integer = 10
    S = real

    function FAK(input n : integer) → integer
    {mengembalikan harga n!, untuk n ≥ 0}

    function POWER(input x : real , input n : integer) → integer
    {mengembalikan harga perpangkatan x^n }

DESKRIPSI
    S ← 0
    for k ← 0 to n do
        S ← S + POWER(x,k) / FAK(k)
    endfor

    return S
```

PROSEDUR atau FUNGSI ?

- Fungsi digunakan bila modul program mengembalikan sebuah nilai
- Prosedur digunakan bila modul program menghasilkan efek netto dari satu atau lebih sekumpulan aksi.
- Dalam praktek perbedaan keduanya tidak jelas, karena prosedur dapat sbg fungsi, fungsi sebagai prosedur

CONTOH FUNGSIMAX

```
function MAKS(input a, b : integer) → integer
{ mengembalikan harga terbesar dari a dan b }

DEKLARASI
{tak ada}
```

DESKRIPSI
    if a ≥ b then
        return a
    else
        return b
    endif

Program Pemanggil:

Algoritma MAKSIMUM
{program utama memanggil fungsi MAKS dari bilangan bulat a dan b, Nilai a dan b dibaca dari piranti masukan}

DEKLARASI
    a, b : integer
    function MAKS(input a, b : integer) → integer
    { mengembalikan harga terbesar dari a dan b }

DESKRIPSI
    read(a,b)
    write('Nilai terbesar : ', MAKS(a,b))

Dapat juga ditulis dengan **prosedur**

Procedure TentukanMAKS(input a, b : integer , output maks : integer)
{ menentukan nilai terbesar dari a dan b dan disimpan di maks }
{ K. Awal : a dan b sudah terdefinisi nilainya }
{ K. Akhir : maks berisi nilai terbesar dari a dan b }

DEKLARASI
{tak ada}

DESKRIPSI
    if a ≥ b then
        maks ← a
    else
        maks ← b
    endif

Program Pemanggil:
Algoritma MAKSIMUM
{program utama memanggil prosedur Tentukan MAKS dari bilangan bulat a dan b, Nilai a dan b dibaca dari piranti masukan}

DEKLARASI

a, b : integer
m : integer

Procedure TentukanMAKS(input a, b : integer , output maks : integer)
{ menentukan nilai terbesar dari a dan b dan disimpan di maks }

DESKRIPSI
read(a,b)
TentukanMAKS(a,b,m)
write('Nilai terbesar : ', m)

PERBEDAAN FUNGSI DAN PROSEDUR

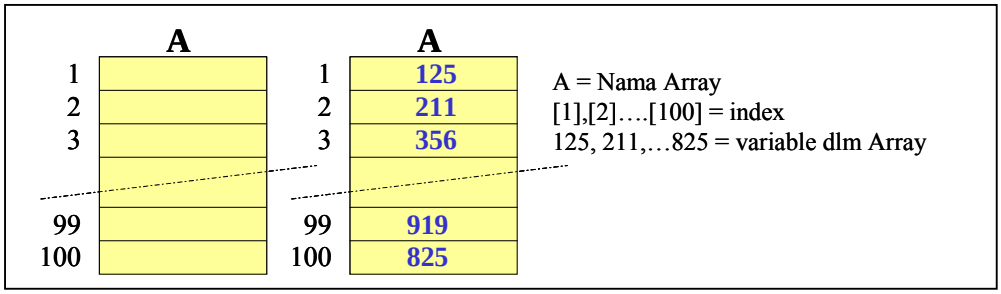
- Fungsi yang mengembalikan sebuah nilai bertipe sederhana dapat ditulis sebagai prosedur dengan *reference parameter (sebagai Output Parameter pada prosedur)*
- Prosedur yang mempunyai sebuah *reference parameter* dapat ditulis sebagai fungsi
- Prosedur yang mempunyai lebih dari sebuah *reference parameter* tidak elegan ditulis sebagai fungsi
- Fungsi sangat tepat digunakan bila nilai fungsi digunakan dalam ekspresi matematika

11. ARRAY

Defisini

- Array (*larik*) adalah sruktur data yang menyimpan sekumpulan elemen yang bertipe sama.
- Setiap elemen diakses langsung melalui *index*nya.
- Index larik haruslah tipe data yang berurutan, seperti *integer* atau *karakter*.

Contoh : A[1], A[2], …. A[100], secara visual dapat digambarkan sebagai berikut.:



Mendefinisikan Array di DEKLARASI

Mendefinisikan berarti memesan sejumlah tempat di memori, yang terdiri dari :

- Banyaknya elemen array
- Tipe elemen array, berupa :
  - Tipe sederhana (*integer, real, char, boolean, string*)
  - Tipe terstruktur (tipe bentukan seperti record)
  - Tipe array

Pendefinisian dalam DEKLARASI :

- Sebagai variabel
- Sebagai Tipe baru
- Ukuran max sebagai konstanta

Contoh mendefinisikan Array (larik) didalam bagian DEKLARASI

1. **Sebagai Variabel**
   Misalkan:
   - L adalah nama variable yang mempunyai 50 buah elemen yang tipe integer. Index array bertipe integer dan dimulai dari 1
   - NAMAMHS adalah variable array yang mempunyai 10 buah elemen yang bertipe string. Index array bertipe char dan dimulai dari ‘a’.
   - NILUJIAN adalah variable array yang mempunyai 75 buah elemen yang bertipe real. Index array bertipe integer dan dimulai dari 0

CONTOH:

```
DEKLARASI
    L : array[1..50] of integer
    NAMAMHS : array['a'..'j'] of string
    NILUJIAN : array[0..74] of real
```

2. Sebagai Tipe Baru

Misalkan:

- Tabint didefinisikan sebagai nama sebuah *tipe baru* untuk larik(array) yang bertipe integer
- Elemen array adalah 100 buah elemen
- Sebuah array integer bernama P dan berukuran 100 elemen dapat didefinisikan bertipe TabInt

CONTOH:

```
DEKLARASI
    Type TabInt : array[1..100] of integer
    P : TabInt
```

3. Mendefinisikan ukuran maximum elemen array sebagai sebuah tetapan

Misalkan:

- TabInt didefinisikan sebagai nama sebuah tipe baru untuk larik(array) yang bertipe integer.
- Banyak elemen array maximum 1000 buah elemen.
- Sebuah array integer yang bernama P dan berukuran max 1000 elemen dapat didefinisikan bertipe TabInt

CONTOH:

```
DEKLARASI
    const Nmaks = 1000
    type TabInt : array[1..Nmaks] of integer

    P : TabInt
```

PEMROSESAN ARRAY

- Elemen array tersusun secara urut.
- Bila index sudah didefinisikan, setiap elemen diproses secara urut melalui index yang urut
- Proses berlangsung mulai dari elemen array yang pertama sampai elemen array yang terakhir.
- Proses adalah aksi yang dilakukan terhadap elemen array berupa pengisian data, read, write atau lainnya.

PEMROSESAN ARRAY

Algoritma Pemrosesan_Array

{memproses setiap elemen array secara beruntun, mulai dari index terkecil sampai index terbesar}

```
DEKLARASI
    const Nmaks = 100 {jumlah elemen array}
    type larik : array[1..Nmaks] of integer

    A : larik
    k : integer {index array}

DESKRIPSI
    for i ← 1 to N do
        PROSES A[i]
    endfor
```

INISIALISASI

```
Procedure Inisialisasi(output A: larik)
{Inisialisasi setiap elemen larik A dengan nol}
{K.Awal : larik A belum terdefinisi nilai elemennya}
{K.Akhir : seluruh elemen larik bernilai nol}

DEKLARASI
    k : integer          {pencatat index larik}

DESKRIPSI
    for k ← 1 to Nmaks do
        A[k] ← 0
    endfor
```

MENGISI ARRAY VIA PIRANTI MASUKAN

```
Procedure Isi_Array(output A:larik)
{mengisi elemen array A[1..Nmaks] dengan nilai yang dibaca dari piranti masukan}
{K.Awal: Array A[1..Nmaks] belum terdefinisi nilai elemennya}
{K.Akhir: Setelah pembacaan, seluruh elemen array A[1..Nmaks] berisi nilai yang
dibaca dari piranti masukan}

DEKLARASI
    k   : integer              {pencatat index array]

DESKRIPSI
    for k ← 1 to Nmaks do
        Read(A[k])
    endfor
```

Sering data yang diisi tidak sama dengan seluruh elemen array yang didefinisikan (Nmaks), banyak elemen array yang dipakai disebut jumlah elemen efektif*, Jumlah elemen efektif disimpan pada variable tertentu (N), (dimana N ≤ Nmaks)*

```
Procedure Isi_Array2(output A:larik, input N:integer)
{mengisi elemen array A[1..N] dengan nilai yang dibaca dari piranti masukan}
{K.Awal: Array A[1..N] belum terdefinisi nilai elemennya, sedang N sudah terisi jumlah
elemen efektifnyai}
{K.Akhir: Setelah pembacaan, sebanyak N buah elemen array, A berisi nilai yang dibaca
dari piranti masukan}

DEKLARASI
    k   : integer                {pencatat index array]

DESKRIPSI
    for k ← 1 to N do
        read(A[k])
    endfor
```

Program Utama Memanggil Isi_Array2

Algoritma Mengisi_Array

{program utama untuk mengisi elemen array dengan nilai yang dibaca dari piranti masukan}

```
DEKLARASI
    const Nmaks = 100        {banyaknya elemen array}
    type larik : array[1..Nmaks] of integer

    A  : larik
    k  : integer       {index array}
    N  : integer       {banyak elemen array yang dipakai ≠ Nmaks , N ≤ Nmaks}
```

Procedure Isi_Array2(output A:larik, input N:integer)

{mengisi elemen array A[1..N] dengan nilai yang dibaca dari piranti masukan}

```
DESKRIPSI
    read(N)
    Isi_Array2(A,N)
```

MENULIS ARRAY VIA PIRANTI KELUARAN

```
Procedure Tulis_Array(input A:larik, input N:integer)
{menyetak elemen array A[1..N] ke piranti keluaran}
{K.Awal: Array A[1..N] sudah terdefinisi nilai elemennya, sedang N sudah berisis ukuran array yang terpakai}
{K.Akhir: Diakhir prosedur, sebanyak N buah elemen array, A tercetak nilainya ke piranti masukan}

DEKLARASI
    k   : integer                {pencatat index array]

DESKRIPSI
    for k ← 1 to N do
        write(A[k])
    endfor
```

MENYIMPAN SEMENTARA DATA

```
Algoritma Without_Array1
{Program tanpa array}

DEKLARASI
    XI  : integer
    i   : integer

DESKRIPSI
    {baca 5 buah nilai integer, simpan di XI}
    for i ← 1 to 5 do
        read(XI)
    endfor

    {cetak setiap nilai XI ke piranti keluaran}
    for i ← 1 to 5 do
        write(XI)
    endfor
```

Bila algoritma dijalankan, dengan nilai nilai yang diisikan ke piranti masukan adalah :

10 200 3000 400 50

maka outputnya adalah

50
50
50
50
50

kenapa 10, 200, 3000, 400 tidak tercetak ?, karena variable menampung 1 nilai, dimana setiap looping, diadakan pengisian baru, otomatis, harga yang terlebih dulu terisi, akan digantikan dengan data yang terakhir.

Dengan data yang sama bandingkan dengan algortima dibawah ini:

Algoritma With_Array1
{Program dengan array}

```
DEKLARASI
    X   : array[1..5] of integer
    i   : integer

DESKRIPSI
    {baca 5 buah nilai integer, simpan di X[1..5] }
    for i ← 1 to 5 do
        read(X[i])
    endfor

    {cetak setiap nilai X[I] ke piranti keluaran}
    for i ← 1 to 5 do
        write(X[i])
    endfor
```

Bila algoritma dijalankan, maka outputnya adalah :

10
200
3000
400
50

MENGHEMAT NAMA-NAMA VARIABEL

Algoritma Without_Array2
{Program tanpa array}

```
DEKLARASI
    X1, X2,X3, X4, X5    : integer
    i    : integer

DESKRIPSI
    {baca 5 buah nilai integer, simpan di X1, X2,... X5}
    read(X1)
    read(X2)
    read(X3)
    read(X4)
    read(X5)

    {cetak  X1, X2,... X5 kepiranti keluaran}
```

```
write(X1)
write(X2)
write(X3)
write(X4)
write(X5)
```

Bagaimana, kalau pengisian sampai 1000 elemen ? Hal ini jelas tidak praktis

Algoritma With_Array2

```
{Program dengan array, tanpa pengulangan}

DEKLARASI
    X   : array[1..5] of integer
    i   : integer

DESKRIPSI
    {baca 5 buah nilai integer, simpan di X[1], X[2],... X[5]}
    read(X[1])
    read(X[2])
    read(X[3])
    read(X[4])
    read(X[5])

    {cetak  X[1], X[2],... X[5] kepiranti keluaran}
    write(X[1])
    write(X[2])
    write(X[3])
    write(X[4])
    write(X[5])
```

Dengan array, menyimpan 5 nilai cukup diperlukan satu variable saja (X), namun X terdiri dari 5 elemen, untuk menyimpan 1000 elemen X juga membutuhkan satu buah variable yang terdiri 1000 elemen, dengan array ditulis satu kali saja didalam struktur pengulangan. Berikut ini algoritma dengan array dengan pengulangan

Algoritma With_Array3

```
{Program dengan array, dengan pengulangan}

DEKLARASI
    X   : array[1..1000] of integer
    i   : integer

DESKRIPSI
    {baca 1000 buah nilai integer, simpan di X[1], X[2],... X[1000]}
    for i ← 1 to 1000 do
        read(X[i])
    endfor

    {cetak setiap nilai X[I] ke piranti keluaran}
```

```
{cetak  X[1], X[2], ... X[1000] kepiranti keluaran}
for i ← 1 to 1000 do
    write(X[i])
endfor
```

ARRAY BERTIPE TERSTRUKTUR

Selain bertipe sederhana, Array juga bertipe terstruktur, Berikut ini contohnya :

Algoritma Baca_Array_Mhs
{Mengisi elemen array Mahasiswa dengan data yang dibaca dari piranti masukan}

```
DEKLARASI
    const  Nmaks  =  100
    type   Mhs    :  record <  NIM      :  integer
                              Nama     :  string
                              KdMata   :  string
                              Nilai    :  char
                           >
    TabMhs : array[1..Nmaks] of Mhs
    k : integer
    N : integer

DESKRIPSI
    read(N)
    for k ← 1 to N do
        read(TabMhs[k].NIM)
        read(TabMhs[k].Nama)
        read(TabMhs[k].KdMata)
        read(TabMhs[k].Nilai)
    endfor
```

Selain bertipe terstruktur, dapat juga bertipe array lain

Algoritma Baca_Array_Mhs
{Mengisi elemen array Mahasiswa dengan data yang dibaca dari piranti masukan}

```
DEKLARASI
    const  Nmaks  =  100
    type   Matakul :  record <  KodeMk   :  string
                                NamaMk   :  string
                                Nilai    :  char
                             >
    type   Mhs    :  record <  NIM      :  integer
                              Nama     :  string
                              MK       :  array[1..4] of MataKul
                             >
    LarikMhs : array[1..Nmaks] of Mhs
    i, j : integer
    N : integer
```

```
DESKRIPSI
    read(N)
    for i ← 1 to N do
        read(LarikMhs[i].NIM)
        read(LarikMhs[i].Nama)
        for j ← 1 to 4 do
            read(LarikMhs[i].MK[j].KodeMK)
            read(LarikMhs[i].MK[j].NamaMK)
            read(LarikMhs[i].MK[j].Nilai)
        endfor
    endfor
```

DUA ATAU LEBIH ARRAY

Dua array dapat dioperasikan sekaligus.

Contoh :

Misalkan nilai Ujian N orang, disimpan dalam array Ujian[1..N]**.** Kemudian hitung nilai index (A/B/C/D/E), mahasiswa menyimpan nilai index didalam Index[1..N]

Deklarasi Data

```
DEKLARASI
    const Nmaks =  200
    type larik_ujian : array [1..Nmaks] of real
    type larik_ujian : array [1..Nmaks] of char
```

Prosedur menghitung nilai Index

```
Procedure Hitung_Index (   input Ujian : larik_ujian,
                           input N : integer,
                           output Indeks : larik_index)
{Menghitung Index Nilai Ujian}
{K.Awal : N sudah berisi ukuran array; elemen array Ujian[1..N] sudah terdefinisi
nilainya}
{K. Akhir : array Indeks[1..N] berisi nilai indeks ujian}

DEKLARASI
    i : integer {index array}

DESKRIPSI
    for i ← 1 to N do
        case (Ujian[i])
              Ujian[i] ≥ 80: Indeks[i] ← 'A'
              70 ≤ Ujian[i] < 80: Indeks[i] ← 'B'
              55 ≤ Ujian[i] < 70: Indeks[i] ← 'C'
```

```
            40 ≤ Ujian[i] < 55: Indeks[i] ← 'D'
            Ujian[i] < 45: Indeks[i] ← 'E'
        endcase
    endfor
```

INISIALISASI ARRAY

- Mengisi harga awal untuk seluruh elemen array, misalkan "mengosongkan" elemen array

MENULIS ARRAY VIA PIRANTI KELUARAN

- Kapan menggunakan Array ?
  - Array dibutuhkan untuk menyimpan sementara data yang bertipe sama dalam memori
  - Menghemat nama-nama variabel
- Array bertipe terstruktur
- Dua atau lebih Array

BAB XII
PENCARIAN
(SEARCHING)

12.1 UMUM

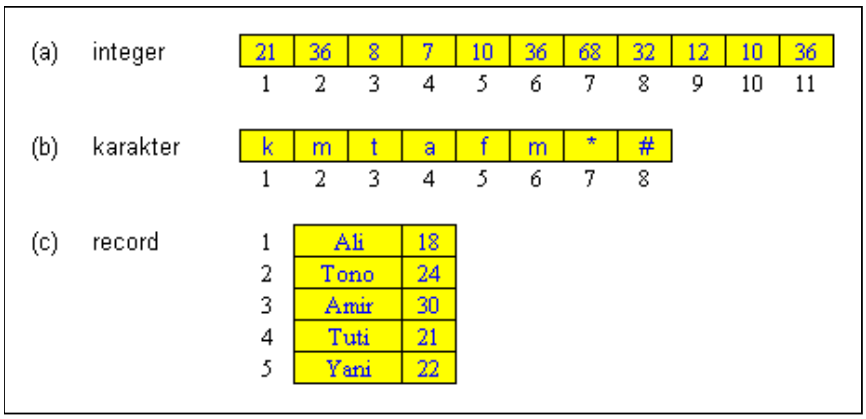
Searching merupakan proses yang utama dalam pemrograman. Proses pencarian adalah mencari / menemukan nilai tertentu dalam sekumpulan data dan mempunyai tipe yang sama (tipe dasar atau tipe bentukan).

Contoh : untuk menghapus atau mengubah nilai dalam suatu data base, langkah pertama yang harus dilakukan adalah mencari apakah nilai yang dicari ada didalam data base tersebut.

Bila ada, nilai tersebut dapat dihapus atau di ubah. Penyisipan data kedalam data base dimulai dengan mencari apakah data yang dimaksud ada dalam data base, bila sudah ada dan mengandaikan tidak boleh ada duplikasi data, maka data tsb. tidak jadi disisipkan, sedang bila belum ada maka sisipkan data tsb. kedalam data base

12.2 ARRAY

Array atau Larik merupakan tipe data terstruktur. Sebuah array dapat dimisalkan sebagai sekumpulan kotak yang menyimpan sekumpulan elemen bertipe sama secara berturutan (*sequential*) di memori utama. Index array haruslah tipe yang mempunyai keterurutan, misal : integer, karakter



Gambar 12.1. Type Array

Untuk (a) : array bertipe integer

(b) : array bertipe karakter ;

(c) : array bertipe record

Definisi Searching dalam Array

Diberikan sebuah array, namakan L, dan X elemen bertipe sama dengan elemen array L. Carilah X terdapat didalam array L *(lihat gambar 1)*

a) Pesan yang muncul bila X ditemukan/tidak

write (X, ‘ditemukan) atau

write (X, ‘tak ditemukan)

b) Index Elemen Array

X = 68 ; IDX = 7 dan

X = 100 ; IDX = 0

c) Boolean

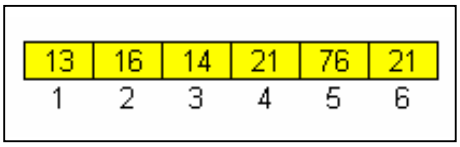
X = 68 ; ketemu = true

X = 100 ; ketemu = false

12.3 SEQUENTIAL SEARCH

Melakukan pencarian secara beruntun, mulai dari elemen pertama sampai terakhir.

Contoh :



Cari 21; **Found !,** Idx = 4

Cari 13; **Found !,** Idx = 1

Cari 30; **Not Found !,** Idx = 0

Pencarian dilakukan mulai Awal (Idx = 1) sampai Akhir (Idx = 6), bila tidak ditemukan Idx berisi angka 0

Berikut ini adalah algoritma untuk pencarian secara sequential (berurutan mulai data awal sampai data terakhir)

FUNGSI PEMANGGILAN

Function CARI_X (input L : larik, input N : integer, input X : integer) → boolean
{ mengembalikan nilai true jika ketemu, sebaliknya false}

DEKLARASI
k : integer *{index larik}*

DESKRIPSI
```
k ← 1
{periksa selama k < N dan L[k] ≠ X}
while ( k < N ) and ( L[k] ≠ X ) do
    k ← k + 1
endwhile
{ k = N or L[k] = X }

if L[k] = X then
    return true
else
    return false
endif
```

Function CARI_IDX (input L : larik, input N : integer, input X : integer) →integer
{ mengembalikan index larik L yg elemennya = X }

DEKLARASI
k : integer *{index larik}*

DESKRIPSI
```
k←1

{periksa selama k < N dan L[k] ≠ X}
while ( k < N ) and ( L[k] ≠ X ) do
    k←k + 1
endwhile
{ k = N or L[k] = X }

if L[k] = X then
    return k
else
    return 0
endif
```

CONTOH Pemanggilan FUNGSI

{misalkan P dan IdxP sudah didefinisikan tipenya didalam bagian DEKLARASI}

DESKRIPSI

```
read (P)
IdxP = CARI_IDX(L,P)

if IdxP = 0 then
    write (P,'tidak ditemukan')
else
    {proses terhadap L[IdxP]}
    .....
endif
```

Cara Pemanggilan FUNGSI

{misalkan P dan IdxP sudah didefinisikan tipenya didalam bagian DEKLARASI}

```
read(P)
if not Cari_X(L,P) then          { Cari_X bernilai false }
    write(P,'tidak ditemukan !!!')
else
    { proses terhadap P }
    …….
endif
```

PROGRAM UTAMA

{misalkan P sudah didefinisikan tipenya didalam Deklarasi}

DESKRIPSI

```
read (P)
if not Cari_X(L,P) then                    {cari_X bernilai false}
    write (P,'tidak ditemukan')
else
    {proses terhadap L[IdxP]}
    .....
endif
```

PROSEDUR

Procedure BACA_LARIK(output L: larik, input N : integer)

{ mengisi elemen larik L[1..N]dari piranti masukan }

{ k. awal : larik A belum terdefinisi elemennya ; N sudah berisi jumlah elemen efektif }

{ k. akhir : setelah pembacaan, N elemen larik A berisi nilai yg dibaca dari piranti masukan}

DEKLARASI

k : integer *{pencatat index larik}*

DESKRIPSI

```
for k ← 1 to N do
    read(A[k])
endfor
{ k = N or L[k] = X }
```

CARI_IDX TYPE 1

Procedure CARI_IDX (input L : larik, input N : integer, input X : integer, output IDX : integer)

{ mencari nilai dalama array/larik L[1..n] }

{ kondisi awal : nilai X dan elemen array/larik L[1..N] sudah terdefinisi }

{ k. akhir : IDX = Index dari larik L tempat X ditemukan; IDX=0, jika X tidak ditemukan }

DEKLARASI

k : integer *{index larik}*

DESKRIPSI

```
k ← 1
while ( k < N ) and ( L[k] ≠ X ) do
    k ← k + 1
endwhile
{ k = N or L[k] = X }

if L[k] = X then                { x ditemukan }
    IDX ← k
else
    IDX ← 0
endif
```

CARI_IDX TYPE 2

```
Procedure CARI_IDX ( input L : larik, input N : integer, input X : integer, output IDX : integer)
{ mencari nilai dalama array/larik L[1..n] }
{ kondisi awal : nilai X dan elemen array/larik L[1..N] sudah terdefinisi }
{ k. akhir : IDX = Index dari larik L tempat X ditemukan; IDX=0, jika X tidak ditemukan }

DEKLARASI
    k :  integer                {index larik}
    ketemu : boolean            {true bila ketemu; false bila tak ketemu}

DESKRIPSI
    k ← 1
    ketemu ← false
    while ( k < N ) and ( not ketemu ) do
        if L[k] = X then
            ketemu ← true
        else
            k ← k + 1
        endif
    endwhile                    { k = N or ketemu }
    if  ketemu  then            { x ditemukan }
        IDX ← k
    else
        IDX ← 0
    endif
```

Algoritma SEARCHING_IDX

```
{ Program mencari nilai tertentu dalam array }
DEKLARASI
    const Nmax = 100                { jumlah maximum elemen larik }
    type Larik : array[1..Nmax] of integer
    A :  larik
    X :  integer                    { elemen yang dicari }
    IDX : integer                   { index larik tempat X ditemukan }
    procedure BACA_LARIK(output L: larik, input N : integer )
    { mengisi elemen larik L[1..N] dari piranti masukan }
    procedure CARI_IDX ( input L : larik, input N : integer, input X : integer, output IDX : intege
    { mencari nilai X di dalam larik L[1.. N] }

DESKRIPSI
    read (N)
    BACA_LARIK(A, N)
    read (X)
    CARI_IDX(A, N, X, IDX)
    if IDX = 0 then             { x ditemukan }
        write ( x, 'tidak ditemukan')
    else
        write ( x, 'ditemukan pada index larik ke ', IDX)
    endif
```

CARI_X TYPE 1

```
Procedure CARI_X ( input L : larik, input N : integer, input X : integer, output ketemu : boolean )
{ mencari nilai dalama array/larik L[1..n] }
{ kondisi awal : nilai X dan elemen array/larik L[1..N] sudah terdefinisi }
{ k. akhir : IDX = Index dari larik L tempat X ditemukan; IDX=0, jika X tidak ditemukan }

DEKLARASI
    k :  integer                    {index larik}

DESKRIPSI
    k ← 1
    while ( k < N ) and ( L[k] ≠ X ) do
        k ← k + 1
    endwhile
    { k = N  or  L[k] = X  }

    if L[k] = X then                { x ditemukan }
        ketemu ← true
    else
        ketemu ← false
    endif
```

CARI_X TYPE2

```
Procedure CARI_X ( input L : larik, input N : integer, input X : integer, output ketemu : boolean)
{ mencari nilai dalama array/larik L[1..n] }
{ kondisi awal : nilai X dan elemen array/larik L[1..N] sudah terdefinisi }
{ k. akhir : IDX = Index dari larik L tempat X ditemukan; IDX=0, jika X tidak ditemukan }

DEKLARASI
    k :  integer                    {index larik}

DESKRIPSI
    k ← 1
    ketemu ← false
    while ( k < N ) and ( not ketemu ) do
        if L[k] = X then
            ketemu ← true
        else
            k ← k + 1               { x ditemukan }
        endif
    endwhile
    { k = N  or  ketemu }
```

Algoritma SEARCHING_X

{ Program mencari nilai tertentu dalam array }

DEKLARASI

```
const Nmax = 100                    { jumlah maximum elemen larik }
type Larik : array[1..Nmax] of integer
A :  larik
X :  integer                        { elemen yang dicari }
found : boolean                     {nilai true bila X ditemukan,sebaliknya false}

procedure BACA_LARIK(output L: larik, input N : integer )
{ mengisi elemen larik L[1..N] dari piranti masukan }
procedure CARI_X ( input L : larik, input N : integer, input X : integer, output ketemu : boolean)
{ mencari nilai X di dalam larik L[1.. N] }
```

DESKRIPSI

```
read (N)
BACA_LARIK(A, N)

read (X)
CARI_X(A, N, X, found)

if found then                       { x ditemukan }
    write ( x, 'tidak ditemukan')
else
    write ( x, 'ditemukan')
endif
```

12.4 BINARY SEARCH

Pencarian BagiDua atau Pencarian Biner (*Binary Search*) adalah metode pencarian yang diterapkan pada sekumpulan Data yang sudah terurut (menaik atau menurun).
Metode ini digunakan untuk pencarian data dengan waktu yang cepat.
Data terurut mutlak diperlukan dalam metode Pencarian BagiDua, berbeda dengan pencarian Beruntun, yang dapat diterapkan baik data terurut maupun data secara acak

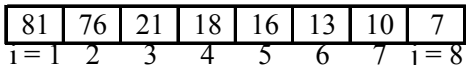
Algoritma BINARY SEARCH
Index kiri adalah ***i*** dan Index kanan adalah ***j*** . Pada awal, i = 1 dan j = N;

Langkah 1 :
Bagi dua elemen larik pada elemen tengah. Elemen Tengah adalah elemen dengan index ***l*** (Elemen Tengah, L[k], membagi Larik menjadi dua bagian, yaitu bagian kiri L[i…k] dan bagian kanan L[k+1..j]

Langkah 2 :
Periksa apakan L[k] = X.
Bila **Benar** , pencarian selesai , sebab X sudah ditemukan
Bila **Salah**, harus ditentukan pencarian ke *kiri* atau ke *kanan*
jika L[k] < X , maka pencarian pada larik *kiri*
jika L[k] > X , maka pencarian pada larik *kanan*

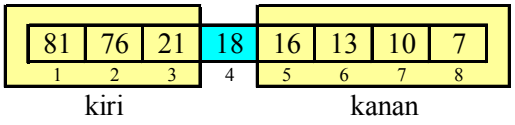
Langkah 3 :
Ulangi langkah 1 sampai X ditemukan atau i > j (yaitu ukuran larik sudah nol)

Ilustrasi pencarian bagidua



(i) Misalkan elemen yang dicari adalah X = 18

Langkah 1 :
i = 1 dan j = 8
Index elemen tengah k = (1 + 8) DIV 2 { warna }



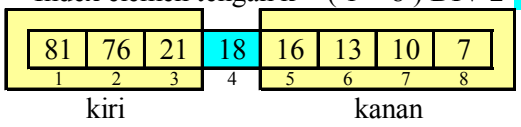
Langkah 2 :
L[4] = 18 ? Ya ! *{ X ditemukan, pencarian dihentikan }*

(ii) Misalkan elemen yang dicari adalah X = 16

Langkah 1 :

i = 1 dan j = 8

Index elemen tengah k = (1 + 8) DIV 2 = { warna }



Langkah 2 :

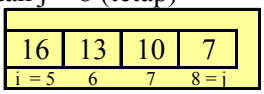
L[4] = 16 ? Tidak !

Harus diputuskan apakah pencarian akan dilakukan di seb. *kiri* atau *kanan* dengan pemeriksaan sbb :

L[4] > 16 ? Ya !

Lakukan pencarian pada larik seb. *kanan* dengan

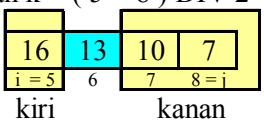
i = k + 1 = 5 dan j = 8 (tetap)



Langkah 1' :

i = 5 dan j = 8

Index elemen tengah k = (5 + 8) DIV 2 = 6



Langkah 2' :

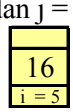
L[6] = 16 ? Tidak !

Harus diputuskan apakah pencarian akan dilakukan di seb. *kiri* atau *kanan* dengan pemeriksaan sbb :

L[6] > 16 ? Tidak !

Lakukan pencarian pada larik seb. *kiri* dengan

i = 5 (tetap) dan j = k - 1 = 5



Langkah 1" :

i = 5 dan j = 5

Index elemen tengah k = (5 + 5) DIV 2 = 5

Langkah 2" :

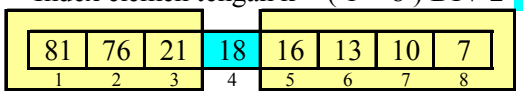
L[5] = 16 ? Ya ! *{ X ditemukan, pencarian dihentikan }*

(iii) Misalkan elemen yang dicari adalah X = 100

Langkah 1 :

i = 1 dan j = 8

Index elemen tengah k = (1 + 8) DIV 2 = { warna }



Langkah 2 :

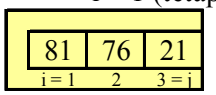
L[4] = 100 ? Tidak !

Harus diputuskan apakah pencarian akan dilakukan di seb. *kiri* atau *kanan* dengan pemeriksaan sbb :

L[4] > 100 ? Tidak !

Lakukan pencarian pada larik seb. *kiri* dengan

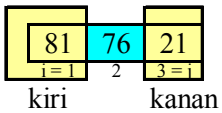
i = 1 (tetap) dan j = k - 1 = 3



Langkah 1' :

i = 1 dan j = 3

Index elemen tengah k = (1 + 3) DIV 2 = 2



Langkah 2' :

L[2] = 100 ? Tidak !

Harus diputuskan apakah pencarian akan dilakukan di seb. *kiri* atau *kanan* dengan pemeriksaan sbb :

L[2] > 100 ? Tidak !

Lakukan pencarian pada larik seb. *kiri* dengan

i = 1 (tetap) dan j = k - 1 = 1

Langkah 1" :

i = 1 dan j = 1

Index elemen tengah k = (1 + 1) DIV 2 = 1

Langkah 2" :

L[1] = 100 ? Tidak *{ X ditemukan, pencarian dihentikan }*
Harus diputuskan apakah pencarian akan dilakukan di seb. *kiri* atau *kanan*
dengan pemeriksaan sbb :
L[1] > 100 ? Tidak !
Lakukan pencarian pada larik seb. *kiri* dengan
i = 1 (tetap) dan j = k - 1 = 0
karena i > j , maka tidak ada lagi bagian larik yang tersisa.
Jadi X tidak ditemukan didalam larik
Pencarian dihentikan

```
Procedure BAGIDUA1 ( input L : larik, input N : integer, input X : integer,
                                output IDX : integer )
{ mencari nilai X didalam larik L[1..N] yang sudah terurut menurun dengan metode
  pencarian BAGIDUA. Output Index IDX yg L[IDX] = X, IDX berharga 0 jika X tidak ditemukan
K. Awal : Larik L[1..N] sudah berisi data yang terurut menurun , dan X adalah nilai yg akan dicari
K. Akhir : IDX berisi index Larik tempat X ditemukan, IDX = 0 jika X tidak ditemukan }

DEKLARASI
    i, j  :  integer          {index kiri dan kanan larik}
    k :      integer          {index elemen tengah}
    ketemu : boolean          {ketemu atau tidak}

DESKRIPSI
    i ← 1
    j ← N
    ketemu ← false
    while ( not ketemu ) and ( i ≤ j ) do
      k ← ( i + j ) div 2
      if ( L[k] = X ) then
            ketemu ← true
      else        { L[k]<> X }
            if (L[k] > X) then
                  {akan lakukan pencarian pada larik kanan, set index ujung kiri larik baru}
                  i ← ( k + 1 )
            else
                  {akan lakukan pencarian pada larik kiri, set index ujung kiri larik baru}
                  j ← ( k - 1 )
            endif
      endif
    endwhile
    { ketemu = N true or i > j }
    if (ketemu) then
      IDx ← k        { X ketemu }
    else
      IDx ← 0        { X tidak ketemu }
    endif
```

```
Procedure BAGIDUA2 ( input L : larik, input N : integer, input X : integer, output IDX : integer )
{ mencari nilai X didalam larik L[1..N] yang sudah terurut menaik dengan metode pencarian
 BAGIDUA. Output Index IDX yg L[IDX] = X, IDX berharga 0 jika X tidak ditemukan
K. Awal : Larik L[1..N] sudah berisi data yang terurut menaik, dan X adalah nilai yg akan dicari
K. Akhir : IDX berisi index Larik tempat X ditemukan, IDX = 0 jika X tidak ditemukan }

DEKLARASI
    i, j :  integer          {index kiri dan kanan larik}
    k :     integer          {index elemen tengah}
    ketemu : boolean         {ketemu atau tidak}

DESKRIPSI
    i ← 1
    j ← N
    ketemu ← false

    while ( not ketemu ) and ( i ≤ j ) do
      k ← ( i + j ) div 2
      if ( L[k] = X ) then
          ketemu ← true
      else       { L[k]<> X }
          if (L[k] < X) then
              {akan lakukan pencarian pada larik kanan, set index ujung kiri larik baru}
              i ← ( k + 1 )
          else
              {akan lakukan pencarian pada larik kiri, set index ujung kiri larik baru}
              j ← ( k - 1 )
          endif
      endif
    endwhile
    { ketemu = N true or i > j }

    if (ketemu) then
      IDx ← k       { X ketemu }
    else
      IDx ← 0       { X tidak ketemu }
    endif
```

12.5 ALGORITMA PENCARIAN PADA LARIK TERSTRUKTUR

DEKLARASI

```
const Nmax = 100
type Mahasiswa : record < NIM    : integer ,    {Nomor Induk Mahasiswa}
                          Nama   : string ,     {Nama Mahasiswa}
                          KdMK   : string ,     {Kode Mata Kuliah }
                          Nilai  : char         {Index Nilai A/B/C}
                        >
type TabMhs : array[1..Nmax] of Mahasiswa
M : MTabMhs
```

(a) Pencarian Beruntun

```
Procedure CARINIM1 ( input M: TabMHS, input N : Integer, Input NIMMhs : integer
                     output Idx : integer )
{mencari keberadaan NIMMhs didalam larik M[1..N] dng metode pencarian beruntun}
{K. Awal : value NIMMhs, N dan elemen larik M[1..N] sudah terdefinisi}
{K. Akhir : IDX berisi index Larik tempat NIMMhs ditemukan, IDX = 0 ,
jika NIMMhs tidak ditemukan }
```

DEKLARASI

```
k       : integer
ketemu  : boolean
```

DESKRIPSI

```
k ← 1
ketemu ← false

while ( not ketemu ) and ( k ≤ N ) do
if    M[k].NIM = NIMMhs then
      ketemu ← true
else
      k ← k + 1
endif
endwhile
{ k > N or ketemu }

if ketemu  then
  IDx ← k      {NIMMhs ditemukan}
else
  IDx ← 0
endif          {NIMMhs tidak ditemukan}
```

(b) Pencarian BagiDua

```
procedure CARINIM2 ( input M: TabMHS, input N : Integer, Input NIMMhs : integer
                     output Idx : integer )
{mencari NIMMhs didalam larik M[1..N] yg sudah terurut menaik berdasarkan NIM dengan
metode pencarian BagiDua. Keluaran Output adalah Index, dimana M{Idx] = NIMMhs
Idx = 0, bila tidak ditemukan }
{K. Awal : Larik M[1..N] sudah terisi data terurut menaik, dan NIMMhas adalah yg dicari}
{K. Akhir : IDX berisi index Larik tempat NIMMhs ditemukan, IDX = 0 ,
jika NIMMhs tidak ditemukan }

DEKLARASI
    k       : integer
    ketemu  : boolean

DESKRIPSI
    i ← 1
    j ← 1
    ketemu ← false

    while ( not ketemu ) and ( i ≤ j ) do
      k ← (i + j) div 2          {bagidua Larik pada posisi k }
      if  M[k].NIM = NIMMhs then
          ketemu ← true
      else
          if  M[k].NIM < NIMMhs then
              {pencarian sebelah kanan, set index ujung kiri larik yg baru }
              i ← k + 1
          else
              {pencarian sebelah kiri, set index ujung kanan larik yg baru }
              j ← k - 1
          endif
      endif
    endwhile
    { i > j or ketemu }

    if ketemu  then {NIMMhs ditemukan}
      IDx ← k
    else
      IDx ← 0       {NIMMhs tidak ditemukan}
    endif
```

12.6 PENCARIAN BERUNTUN ATAU BAGIDUA

12.6 PAKAI PENCARIAN BERUNTUN ATAU BAGIDUA

Perbandingan antara Algoritma *Beruntun* atau *BagiDua*

(a) Untuk larik ukuran 256 elemen
- Untuk *Beruntun* melakukan pembandingan elemen larik sebanyak 256 kali
- Untuk *BagiDua* melakukan pembandingan elemen larik sebanyak $^2\log(256) = 8$ kali

(b) Untuk larik ukuran 1024 elemen
- Untuk *Beruntun* melakukan pembandingan elemen larik sebanyak 1024 kali
- Untuk *BagiDua* melakukan pembandingan elemen larik sebanyak $^2\log(1024) = 10$ kali

(c) Untuk larik ukuran N elemen
- Untuk *Beruntun* melakukan pembandingan elemen larik sebanyak N kali
- Untuk *BagiDua* melakukan pembandingan elemen larik sebanyak $^2\log(N)$ kali

karena $^2\log(N) < N$ untuk N yang besar, maka algoritma BagiDua *lebih cepat* dari pada Algoritma pencarian Beruntun, apabila datanya tersusun urut (menaik atau menurun), tetapi bila data tidak tersusun urut, yang dapat dipakai hanyalah yang beruntun

BAB XIII
PENGURUTAN
(SORTING)

13.1 DEFINISI PENGURUTAN

Pengurutan *menaik* berarti menyusun elemen larik sedemikian rupa sehingga

$L[1] \leq L[2] \leq L[3] \leq \ldots\ldots \leq L[N]$

Pengurutan *menurun* berarti menyusun elemen larik sedemikian rupa sehingga

$L[1] \geq L[2] \geq L[3] \geq \ldots\ldots \geq L[N]$

Data yang diurutkan dapat bertipe Dasar ataupun tipe Rekaman
Jika bertipe rekaman, maka *field* yang harus diurutkan
Field yang dijadikan dasar pengurutan dikenal dengan *key field*

Contoh :

i.	23, 28, 40 ,61, 65, 65, 90, 100	{data *integer* terurut *menaik* }
ii.	50.4, 23.6, 15.14, 0.01, -53.5, -200.23	{data *real* terurut *menurun* }
iii.	Albert, Buddy, Charly, Donny, Dicky,	{data *string* terurut *menaik* }
iv.	<0420201, Sherina >, <04020208, Sandeep>, <0420210, Prasanna >, <040214, Eugene>	{data mahasiswa *menaik* berdasarkan *field NIM* }

Keuntungan data terurut [baik *menaik(Ascending)* atau *menurun(Desending)*]

1 Mempercepat pencarian (Metode BagiDua)
2 Harga Max dan Min dapat langsung diketahui
{bila *Ascending* : Min = elemen Pertama ; Max = elemen terakhir }
{bila *Descending* : Max = elemen Pertama ; Min = elemen terakhir }

Contoh kehidupan sehari-hari
Buku Telephone
Kamus
dll

13.2 PENGURUTAN *Internal* dan *External*

a) *Pengurutan Internal*

Pengurutan terhadap sekumpulan Data yang disimpan didalam memori utama komputer. Umumnya struktur internal yang dipakai untuk pengurutan internal adalah ARRAY/LARIK, sehingga pengurutan Internal disebut juga *pengurutan ARRAY/LARIK*

b) *Pengurutan External*

Pengurutan Data yang disimpan didalam memori sekunder, biasanya data bervolume besar sehingga tidak mampu dimuat semuanya didalam memori komputer, struktur external yang dipakai adalah FILE/ARSIP, sehingga pengurutan External di sebut juga *pengurutan ARSIP*

Karena akses memori utama lebih cepat daripada memori sekunder, maka pengurutan internal lebih cepat dari pada pengurutan External, karena pengurutan External diperlukan *overhead* atau ongkos tambahan untuk mengakses data di media penyimpanan sekunder.
Akan tetapi pengurutan bila dilakukan dalam memori bersifat sementara, dan hilang bila komputer dimatikan

Pembahasan Algoritma adalah sebagai berikut :

a) *Bubble Sort* {Algoritma Pengurutan Gelembung }
b) *Selection Sort* {Algoritma Pengurutan Pilih }
c) *Insertion Sort* {Algoritma Pengurutan Sisip }

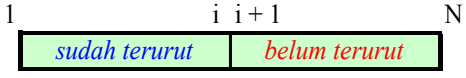
Untuk ketiga Algoritma diatas, menggunakan tipe larik sebagai berikut :

```
DEKLARASI
    const Nmax =  {jumlah maximum elemen larik}
    type Larik = Array[1..Nmax] of integer
```

13.3 PENGURUTAN GELEMBUNG (BUBBLE SORT)

Metode pengurutan Gelembung {Bubble Sort} diinspirasi oleh gelembung sabun yang berada diatas permukaan air. Karena berat jenis gelembung lebih ringan dari pada berat jenis air, maka gelembung selalu terapung keatas permukaan.
Secara umum, **benda berat** akan **terbenam** dan **benda ringan** akan **terapung**

Prinsip pengapungan digunakan pada pengurutan gelembung. Bila menghendaki larik terurut *menaik* , maka elemen larik yang berharga paling kecil *diapungkan* , artinya diangkat ke atas (atau keujung kiri larik) melalui proses pertukaran. Proses pengapungan ini dilakukan sebanyak N - 1 langkah (*pass*) dengan N adalah ukuran larik.
Pada setiap langkah ke-i, larik L[1..N] terdiri dari dua bagian yaitu bagian yang sudah terurut, yaitu L[1..i], dan bagian yang belum terurut L[i + 1..N]
Setelah langkah terakhir, diperoleh Larik[1..N] yang terurut *menaik*



13.3.1 ALGORITMA PENGURUTAN GELEMBUNG

Untuk mendapatkan larik yang terurut menaik (*Ascending*), Algoritma ditulis secara Global sbb :

Untuk setiap *pass* ke-I = 1, 2, 3, …… , N-1, lakukan :
Mulai dari elemen k = N, N-1, ……, i +1, lakukanlah :
1.1 Bandingkan L[k] dengan L[k-1]
1.2 Tukarkan L[k] dengan L[k-1], jika L[k] < L[k-1]
Rincian setiap *pass* adalah sebagai berikut :

Pass 1
Mulai dari elemen k = N, N-1, ……., 2
Bandingkan L[k] dengan L[k-1]
Jika L[k] < L[k-1], maka tukarkan L[k] dengan L[k-1]
Pada akhir langkah 1, elemen L[1] berisi harga minimum Pertama

Pass 2
Mulai dari elemen k = N, N-1, ……., 3
Bandingkan L[k] dengan L[k-1]
Jika L[k] < L[k-1], maka tukarkan L[k] dengan L[k-1]
Pada akhir langkah 2, elemen L[2] berisi harga minimum Kedua
dan larik L[1..2] terurut, sedangkan L[3..N] belum terurut

Pass 3
Mulai dari elemen k = N, N-1, ……., 4
Bandingkan L[k] dengan L[k-1]
Jika L[k] < L[k-1], maka tukarkan L[k] dengan L[k-1]
Pada akhir langkah 3, elemen L[3] berisi harga minimum Kedua
dan larik L[1..3] terurut, sedangkan L[4..N] belum terurut

Pass N-1
Mulai dari elemen k = N
Bandingkan L[k] dengan L[k-1]
Jika L[k] < L[k-1], maka tukarkan L[k] dengan L[k-1]
Pada akhir langkah N-1, elemen L[N-1] berisi harga minimum ke-(N-1)
dan larik L[1..N-1] terurut, sedangkan L[N] tak perlu diurutkan karena
hanya satu-satunya

Contoh

Tinjau Larik L dengan N = 6 buah elemen di bawah ini yang belum terurut.
Larik ini akan diurut Naik.

25	27	10	8	76	21
1	2	3	4	5	6

Pass 1	K = N = 6					21	76
	K = 5				8	21	76
	K = 4			8	10	21	76
	K = 3		8	27	10	21	76
	K = 2	8	25	27	10	21	76

Hasil Akhir Langkah 1

8	25	27	10	21	76
1	2	3	4	5	6

Pass 2 K = N = 6 21 76
K = 5 10 21 76
K = 4 10 27 21 76
K = 3 10 25 27 21 76

Hasil Akhir Langkah 2

8	10	25	27	21	76
1	2	3	4	5	6

Pass 3 K = N = 6 21 76
K = 5 21 27 76
K = 4 21 25 27 76

Hasil Akhir Langkah 3

8	10	21	25	27	76
1	2	3	4	5	6

Pass 4 K = N = 6 27 76
K = 5 25 27 76

Hasil Akhir Langkah 4

8	10	21	25	27	76
1	2	3	4	5	6

Pass 5 K = N = 6 27 76

Hasil Akhir Langkah 5

8	10	21	25	27	76
1	2	3	4	5	6

finish

PROCEDURE

Procedure Tukar(Input/Output a : Integer, Input/Ouput b:Integer)

{menukarkan nilai a dan b }

DEKLARASI

temp : integer

DESKRIPSI

temp ← a
a ← b
b ← Temp

Procedure UrutGelembung1 (Input/Output L:Larik, Input N : Integer)

{Urutkan Larik L[1..N] sehingga terurut Menaik dengan metode BUBBLE SORT}
{K. Awal : Elemen Larik sudah terdefinisi nilai-nilainya}
{K. Akhir : Elemen Larik terurut menarik sedemikian sehingga L[1] ≤ L[2] ≤ L[3] ≤ ....L[N].}

DEKLARASI

```
i     : integer      {pencacah, untuk jumlah langka}
k     : integer      {pencacah, untuk pengapungan pada setiap langkah}
procedure Tukar(Input/output a : integer, input/ouput b:integer)
```

DESKRIPSI

```
for  i ← 1 to N - 1 do
     for  k ← N downto i +1 do
          if  L[k] < L[k-1] then
              Tukar(L[k],L[k-1])
          endif
     endfor
endfor
```

Procedure UrutGelembung2 (Input/Output L:Larik, Input N : Integer)

{Urutkan Larik L[1..N] sehingga terurut Menurun dengan metode BUBBLE SORT}
{K. Awal : Elemen Larik sudah terdefinisi nilai-nilainya}
{K. Akhir : Elemen Larik terurut menarik sedemikian sehingga L[1] ≥ L[2] ≥ L[3] ≥ ....L[N].}

DEKLARASI

```
i     : integer      {pencacah, untuk jumlah langka}
k     : integer      {pencacah, untuk pengapungan pada setiap langkah}
procedure Tukar(Input/Output a : Integer, Input/Ouput b:Integer)
```

DESKRIPSI

```
for  i ← 1 to N - 1 do
     for  k ← N downto i +1 do
          if  L[k] > L[k-1] then
              Tukar(L[k],L[k-1])
          endif
     endfor
endfor
```

13.4 PENGURUTAN PILIH
{Selection Sort}

Metode pengurutan Pilih {*Selection Sort*}, ide dasarnya dari memilih elemen maximum / min dari larik, kemudian menempatkan elemen max/min pada awal atau akhir larik (elemen terujung)

Selanjutnya elemen terujung tersebut di"*isolasi*" dan tidak disertakan pada proses berikutnya.

Proses yang sama dilakukan untuk elemen larik yang tersisa, yaitu memilih elemen max/min berikutnya dan menukarkan dengan elemen terujung larik sisa.

Proses memilih nilai max/min dilakukan pada setiap pass. Jika Larik berukuran N maka jumlah *pass* adalah N-1

Algoritma pengurutan PILIH mempunyai 2 (dua) macam varian yakni :
a) Algoritma Pengurutan Maximum, elemen maximum sebagai dasar pengurutan
b) Algoritma Pengurutan Minimum, elemen minimum sebagai dasar pengurutan

13.4.1 ALGORITMA PENGURUTAN MAXIMUM

Untuk mendapatkan larik yang terurut menaik (*Ascending*), Algoritma ditulis secara Global sbb :

1 JumlahPass = N - 1
2 Untuk setiap *pass* ke-i = 1, 2, ……., JumlahPass
  2.1 cari elemen terbesar (max) mulai elemen ke i sampai elemen ke N
  2.2 Tukarkan max dengan elemen ke-i
  2.3 kurangi N dengan satu

Rincian setiap *pass* adalah sebagai berikut :

Pass 1
Cari elemen max dalam L[1..N]
Tukarkan elemen Maximum dengan elemen L[N]

Pass 2
Cari elemen max dalam L[1..N-1]
Tukarkan elemen Maximum dengan elemen L[N-1]

Pass 3
Cari elemen max dalam L[1..N-2]
Tukarkan elemen Maximum dengan elemen L[N-2]

Pass N-1
Cari elemen max dalam L[1..2]
Tukarkan elemen Maximum dengan elemen L[2]

(elemen yang tersisa adalah L[1], tidak perlu diurut karena hanya satu-satunya)

Contoh

Tinjau Larik L dengan N = 6 buah elemen di bawah ini yang belum terurut.
Larik ini akan diurut Naik.

29	27	10	8	76	21
1	2	3	4	5	6

Pass 1 Cari elemen max didalam larik L[1..6]
Max = L[5] = 76
tukar Max dengan L[N] ; N = 6 , diperoleh

29	27	10	8	21	76
1	2	3	4	5	6

Pass 2 Cari elemen max didalam larik L[1..5]
Max = L[1] = 29
tukar Max dengan L[N-1] ; N = 5, diperoleh

21	27	10	8	29	76
1	2	3	4	5	6

Pass 3 Cari elemen max didalam larik L[1..4]
Max = L[2] = 27
tukar Max dengan L[N-2] ; N = 4, diperoleh

21	8	10	27	29	76
1	2	3	4	5	6

Pass 4 Cari elemen max didalam larik L[1..3]
Max = L[1] = 21
tukar Max dengan L[N-3] ; N = 3, diperoleh

10	8	21	27	29	76
1	2	3	4	5	6

Pass 5 Cari elemen max didalam larik L[1..2]
Max = L[1] = 10
tukar Max dengan L[N-4] ; N = 2, diperoleh

8	10	21	27	29	76
1	2	3	4	5	6

Selesai. Larik L sudah terurut

13.4.2 ALGORITMA PENGURUTAN MINIMUM

Untuk mendapatkan larik yang terurut menaik (*Ascending*), Algoritma ditulis secara Global sbb :
Setiap *pass* ke-i = 1, 2, ……., JumlahPass
1 cari elemen terkecil (min) mulai elemen ke i sampai elemen ke N
2 Tukarkan min dengan elemen ke-i

Rincian setiap *pass* adalah sebagai berikut :

Pass 1

Cari elemen min dalam L[1..N]
Tukarkan elemen Minimum dengan elemen L[1]

Pass 2

Cari elemen min dalam L[2..N]
Tukarkan elemen Minimum dengan elemen L[2]

Pass 3

Cari elemen min dalam L[3..N]
Tukarkan elemen Minimum dengan elemen L[3]

Pass N-1

Cari elemen min dalam L[N-1..N]
Tukarkan elemen Minimum dengan elemen L[N-1]

(elemen yang tersisa adalah L[N], tidak perlu diurut karena hanya satu-satunya)

Contoh

Tinjau Larik L dengan N = 6 buah elemen di bawah ini yang belum terurut.
Larik ini akan diurut Naik.

29	27	10	8	76	21
1	2	3	4	5	6

Pass 1 Cari elemen min didalam larik L[1..6]
Min = L[4] = 8
tukar Min dengan L[1] , diperoleh

8	27	10	29	76	21
1	2	3	4	5	6

Pass 2 Cari elemen min didalam larik L[2..6]
Min = L[3] = 10
tukar Min dengan L[2], diperoleh

8	10	27	29	76	21
1	2	3	4	5	6

Pass 3 Cari elemen min didalam larik L[3..6]
Min = L[6] = 21
tukar Min dengan L[3], diperoleh

8	10	21	29	76	27
1	2	3	4	5	6

Pass 4 Cari elemen min didalam larik L[4..6]
Min = L[6] = 27
tukar Min dengan L[4] , diperoleh

8	10	21	27	76	29
1	2	3	4	5	6

Pass 5 Cari elemen min didalam larik L[5..6]
Min = L[6] = 29
tukar Min dengan L[5], diperoleh

8	10	21	27	29	76
1	2	3	4	5	6

Selesai. Larik L sudah terurut

Procedure Tukar(Input/Output a : Integer, Input/Ouput b:Integer)
{menukarkan nilai a dan b }
DEKLARASI
temp : integer

DESKRIPSI
temp ← a
a ← b
b ← Temp

Procedure UrutMax1 (Input/Output L:Larik, Input N : Integer)
{Urutkan Larik L[1..N] sehingga terurut Menaik dengan metode Selection Maximum}
{K. Awal : Elemen Larik sudah terdefinisi nilai-nilainya}
{K. Akhir : Elemen Larik terurut menarik sedemikian sehingga L[1] ≤ L[2] ≤ L[3] ≤ ....L[N].}

DEKLARASI
i, k : integer *{pencacah, untuk jumlah langka}*
Imax : integer *{pencacah, untuk pengapungan pada setiap langkah}*
procedure Tukar(Input/output a : integer, input/ouput b:integer)

DESKRIPSI
```
for  i ← N downto 2 do
     Imax ← 1
     for  k ← 2 to i  do
          if  L[k] > L[Imax] then
              Imax ← k
          endif
     endfor
     Tukar(L[Imax], L[i] )
endfor
```

Procedure UrutMax2 (Input/Output L:Larik, Input N : Integer)
{Urutkan Larik L[1..N] sehingga terurut Menurun dengan metode Selection Max }
{K. Awal : Elemen Larik sudah terdefinisi nilai-nilainya}
{K. Akhir : Elemen Larik terurut menarik sedemikian sehingga L[1] ≥ L[2] ≥ L[3] ≥ ....L[N].}

DEKLARASI

```
i, k  : integer        {pencacah, untuk jumlah langka}
Imax : integer         {pencacah, untuk pengapungan pada setiap langkah}
procedure Tukar(Input/output a : integer, input/ouput b:integer)
```

DESKRIPSI

```
for  i ← 1 to N do
     Imax ← 1
     for  k ← i + 1 to N  do
          if  L[k] > L[Imax] then
              Imax ← k
          endif
     endfor
     Tukar(L[Imax], L[i] )
endfor
```

```
Procedure UrutMin1 (Input/Output L:Larik, Input N : Integer)
    {Urutkan Larik L[1..N] sehingga terurut Menaik dengan metode Selection Minimum}
    {K. Awal : Elemen Larik sudah terdefinisi nilai-nilainya}
    {K. Akhir : Elemen Larik terurut menarik sedemikian sehingga L[1] ≤ L[2] ≤ L[3] ≤ ....L[N].}
```

DEKLARASI

```
i, k  : integer        {pencacah, untuk jumlah langka}
Imin : integer         {pencacah, untuk pengapungan pada setiap langkah}

procedure Tukar(Input/output a : integer, input/ouput b:integer)
```

DESKRIPSI

```
for  i ← 1 to N-1 do
     Imin ← i
     for  k ← i + 1 to N  do
          if  L[k] < L[Imin] then
              Imin ← k
          endif
     endfor
     Tukar(L[Imin], L[i] )
endfor
```

Procedure UrutMin2 (Input/Output L:Larik, Input N : Integer)

{Urutkan Larik L[1..N] sehingga terurut Menurun dengan metode Selection Min }
{K. Awal : Elemen Larik sudah terdefinisi nilai-nilainya}
{K. Akhir : Elemen Larik terurut menarik sedemikian sehingga L[1] ≥ L[2] ≥ L[3] ≥ ....L[N].}

DEKLARASI

```
i, k  : integer        {pencacah, untuk jumlah langka}
Imin : integer         {pencacah, untuk pengapungan pada setiap langkah}
procedure Tukar(Input/output a : integer, input/ouput b:integer)
```

DESKRIPSI

```
for   i ← N downto 2 do
      Imin ← 1
      for   k ← 2 to i  do
            if   L[k] < L[Imin] then
                 Imin ← k
            endif
      endfor
      Tukar(L[Imin], L[i] )
endfor
```

13.5 PENGURUTAN SISIP

{Insertion Sort}

Metode pengurutan Sisip {*Insertion Sort* }, adalah metode pengurutan dengan cara menyisipkan elemen larik pada posisi yang tepat. Pencarian posisi yang tepat dilakukan dengan melakukan pencarian beruntun didalam larik. Selama pencarian posisi yang tepat dilakukan pergeseran elemen larik.

Algoritma pengurutan ini tepat untuk masalah menyisipkan elemen baru kedalam sekumpulan elemen yang sudah terurut

13.5.1 ALGORITMA PENGURUTAN SISIP

Untuk mendapatkan larik yang terurut menaik (*Ascending*), Algoritma ditulis secara Global sbb :

Untuk setiap *pass* ke-i =2, 3, ……., N lakukan :
1 x ← L[i]
2 sisipkan x pada tempat yang sesuai antara L[1] …. L[i]

Rincian setiap *pass* adalah sebagai berikut :

Andaian (*pass* 1) : L[1] dianggap sudah pada tempatnya

Pass 2

x = L[2] harus dicari tempatnya yang tepat pada L[1..2] dengan cara menggeser elemen L[1..1] ke kanan (atau ke bawah, jika larik vertikal)
bila L[1..1] lebih besar dari L[2]
Misalkan posisi yang tepat adalah k
sisipkan L[2] pada L[k]

Pass 3

x = L[3] harus dicari tempatnya yang tepat pada L[1..3] dengan cara menggeser elemen L[1..2] ke kanan (atau ke bawah, jika larik vertikal)
bila L[1..2] lebih besar dari L[3]
Misalkan posisi yang tepat adalah k
sisipkan L[3] pada L[k]

Pass N

x = L[N] harus dicari tempatnya yang tepat pada L[1..N] dengan cara menggeser
elemen L[1..N-1] ke kanan (atau ke bawah, jika larik vertikal)
bila L[1..N-1] lebih besar dari L[N]
Misalkan posisi yang tepat adalah k
sisipkan L[N] pada L[k]

Hasil dari pass N : Larik L[1..N] sudah terurut, yaitu L[1] ≤ L[2] ≤ ……..≤ L[N]

Contoh

Tinjau Larik L dengan N = 6 buah elemen di bawah ini yang belum terurut.
Larik ini akan diurut Naik.

29	27	10	8	76	21
1	2	3	4	5	6

Andaian (*pass* 1)

Elemen x = L[1] dianggap sudah terurut

29	27	10	8	76	21
1	2	3	4	5	6

Pass 2 Cari posisi yang tepat untuk x = L[2] pada larik L[1..2], diperoleh :

27	29	10	8	76	21
1	2	3	4	5	6

Pass 3 Cari posisi yang tepat untuk x = L[3] pada larik L[1..3], diperoleh :

10	27	29	8	76	21
1	2	3	4	5	6

Pass 4 Cari posisi yang tepat untuk x = L[4] pada larik L[1..4], diperoleh :

8	10	27	29	76	21
1	2	3	4	5	6

Pass 5 Cari posisi yang tepat untuk x = L[5] pada larik L[1..5], diperoleh :

8	10	27	29	76	21
1	2	3	4	5	6

Pass 6 Cari posisi yang tepat untuk x = L[6] pada larik L[1..6], diperoleh :

8	10	21	27	29	76
1	2	3	4	5	6

Selesai. Larik L sudah terurut

Andaian (*pass* 1)

Elemen x = L[1] dianggap sudah terurut

29	27	10	8	76	21
1	2	3	4	5	6

Pass 2 Cari posisi yang tepat untuk x = L[2] pada larik L[1..2], diperoleh :

27	29	10	8	76	21
1	2	3	4	5	6

Pass 3 Cari posisi yang tepat untuk x = L[3] pada larik L[1..3], diperoleh :

10	27	29	8	76	21
1	2	3	4	5	6

Pass 4 Cari posisi yang tepat untuk x = L[4] pada larik L[1..4], diperoleh :

8	10	27	29	76	21
1	2	3	4	5	6

Pass 5 Cari posisi yang tepat untuk x = L[5] pada larik L[1..5], diperoleh :

8	10	27	29	76	21
1	2	3	4	5	6

Pass 6 Cari posisi yang tepat untuk x = L[6] pada larik L[1..6], diperoleh :

8	10	21	27	29	76
1	2	3	4	5	6

Selesai. Larik L sudah terurut

Procedure UrutSisip1 (Input/Output L:Larik, Input N : Integer)

{Urutkan Larik L[1..N] sehingga terurut Menaik dengan metode pengurutan Sisip}
{K. Awal : Elemen Larik sudah terdefinisi nilai-nilainya}
{K. Akhir : Elemen Larik terurut menarik sedemikian sehingga L[1] ≤ L[2] ≤ L[3] ≤ ....L[N].}

DEKLARASI

```
i, j   : integer      {pencacah pass}
x      : integer      {pencacah, untuk pengapungan pada setiap langkah}
ketemu : boolean
```

DESKRIPSI

```
{ elemen L[1] dianggap sudah terurut }
for  i ← 2 to N do      { sisipkan L[i] ke dalam bagian yang sudah terurut }
     x ← L[ i ]         { cari posisi yang tepat untuk x didalam L[1..i-1] sambil menggeser }
     j ← i - 1
     ketemu ← false
     while ( j ≥ 1 ) and (not ketemu) do
           if  x < L[j] then
```

```
                L[j+1] ← L[j]        {geser}
                j ← j - 1
            else
                ketemu ← true
            endif
        endwhile        { j < 1 or ketemu }
        L[j+1] ← x      { sisipkan x pada tempat yang sesuai }
    endfor
```

Procedure UrutSisip2 (Input/Output L:Larik, Input N : Integer)

```
    {Urutkan Larik L[1..N] sehingga terurut Menurun dengan metode Pengurutan Sisip }
    {K. Awal : Elemen Larik sudah terdefinisi nilai-nilainya}
    {K. Akhir : Elemen Larik terurut menarik sedemikian sehingga L[1] ≥ L[2] ≥ L[3] ≥ ....L[N].}
DEKLARASI
    i, j   : integer        {pencacah pass}
    x      : integer        {pencacah, untuk pengapungan pada setiap langkah}
    ketemu : boolean
DESKRIPSI
    { elemen L[1] dianggap sudah terurut }
    for  i ← 2 to N do      { sisipkan L[i] ke dalam bagian yang sudah terurut }
        x ← L[ i ]          { cari posisi yang tepat untuk x didalam L[1..i-1] sambil menggeser }
        j ← i - 1
        ketemu ← false
        while ( j ≥ 1 ) and (not ketemu) do
            if  x > L[j] then
                L[j+1] ← L[j]        {geser}
                j ← j - 1
            else
                ketemu ← true
            endif
        endwhile        { j < 1 or ketemu }
        L[j+1] ← x      { sisipkan x pada tempat yang sesuai }
    endfor
```

13.6 PENGABUNGAN DUA BUAH LARIK TERURUT

{Insertion Sort}

Misalkan ada dua larik L1 dan L2, masing-masing sudah terurut menaik.
Kemudian akan membentuk sebuah larik baru, L3, yang merupakan gabungan dari dua buah larik, sehingga L3 juga terurut menaik.

Contoh

LARIK L1

1	13	24

LARIK L2

2	15	27	30

LARIK L1 & L2

1	2	13	15	24	27	30

Penggabungan dilakukan dengan cara membandingkan satu elemen larik L1 dengan satu elemen Larik L2. Jika elemen L1 lebih kecil dari elemen L2, maka salin L1 ke L3. Elemen berikutnya pada L1 maju satu elemen, sedangkan elemen L2 tetap.

Hal yang sama juga berlaku bila elemen L2 lebih kecil dari L1, maka salin L2 ke L3, elemen berikutnya pada L2 maju satu elemen, sedangkan elemen L1 tetap dengan cara tersebut, akan ada elemen yang sudah habis duluan disalin ke L3, sedang yang lain masih tersisa, kemudian salin seluruh elemen yang tersisa ke L3

Hal yang perlu diperhatikan adalah ukuran Larik L3. Jumlah elemen larik L3 adalah banyak elemen L1 + L2, tetapi jumlah elemen larik L3 harus lebih kecil dari Nmax

L1	L2		L3
1 13 24	2 15 27 30	1 < 2 ® 1	1
1 13 24	2 15 27 30	1 < 2 ® 1	1
1 13 24	2 15 27 30	2 < 13 ® 2	1 2
1 13 24	2 15 27 30	13 < 15 ® 13	1 2 13
1 13 24	2 15 27 30	15 < 24 ® 15	1 2 13 15
1 13 24	2 15 27 30	24 < 27 ® 24	1 2 13 15 24
1 13 24	2 15 27 30	27 ® 27	1 2 13 15 24 27
1 13 24	2 15 27 30	30 ® 30	1 2 13 15 24 27 30

procedure GabLarikUrut (input L1:Larik, input N1: integer,
input L2:Larik, input N2: integer,
output L3:Larik, output N3: integer)

{Penggabungan dua buah Larik terurut, L1 dan L2, menghasilkan larik L3 yg urut menaik}
{K. Awal : Larik L1[1..N1] sudah terdefinisi nilai-nilainya dan urut menaik
Larik L2[1..N2] sudah terdefinisi nilai-nilainya dan urut menaik }
{K. Akhir : Larik L3[1..N3] merupakan hasil gabungan L1 dan L2, dengan urut menaik
N3 = N1 + N2 adalah ukuran larik L3 }

DEKLARASI

k1, k2, k3 : integer

DESKRIPSI

N3 ← N1 + N2
k1 ← 1
k2 ← 1

```
k3 ← 1
while (k1 ≤ N1) and (k2 ≤ N2) do
    if   L1[k1] ≤ L2[k2] then
        L3[k3] ← L1[k1]
        k1 ← k1 + 1              {L1 maju satu elemen}
    else
        L3[k3] ← L2[k2]
        k2 ← k2 + 1              {L2 maju satu elemen}
    endif
    k3 ← k3 + 1
endwhile        { k1 > N1  or  k2 > N2 }
{ salin sisa L1, jika ada}
while (k1 ≤ N1) do
    L3[k3] ← L1[k1]
    k1 ← k1 + 1
    k3 ← k3 + 1
endwhile        { k1 > N1}
{ salin sisa L2, jika ada}
while (k2 ≤ N2) do
    L3[k3] ← L2[k2]
    k2 ← k2 + 1
    k3 ← k3 + 1
endwhile        { k2 > N2 }
```

DEKLARASI

```
k1, k2, k3 : integer
```

DESKRIPSI

```
{ elemen L[1] dianggap sudah terurut }
for  i ← 2 to N do        { sisipkan L[i] ke dalam bagian yang sudah terurut }
    x ← L[ i ]            { cari posisi yang tepat untuk x didalam L[1..i-1] sambil menggese
    j ← i - 1
    ketemu ← false
    while ( j ≥ 1 ) and (not ketemu) do
        if  x > L[j] then
            L[j+1] ← L[j]        {geser}
            j ← j - 1
        else
            ketemu ← true
        endif
    endwhile        { j < 1  or  ketemu }
    L[j+1] ← x      { sisipkan x pada tempat yang sesuai }
endfor
```

BAB XIV
MATRIX

14.1. UMUM

Elemen tipe terstruktur seperti larik (*array*) dapat distrukturkan lagi. Sebuah larik yang setiap elemennya adalah larik lagi disebut MATRIX. MATRIX sudah dikenal secara luas dalam berbagai bidang ilmu, terutama Matematika
Matrix identitas adalah contoh matrix yang dikenal secara umum, karena semua elemen diagonalnya = 1 dan elemen lainnya = 0

$$\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Sebuah sistem persaman yang terdiri dari empat persamaan berikut :

$$2x_1 - 6x_2 + 4x_3 + 12x_4 = -10$$
$$3x_1 + 2x_2 - 8x_3 - 8x_4 = 12$$
$$x_1 + 10x_2 + 8x_3 - 7x_4 = 8$$
$$4x_1 - 5x_2 - 9x_3 + 11x_4 = 21$$

Lazim ditulis dalam bentuk persamaan Matrix $Ax = b$,
dimana :

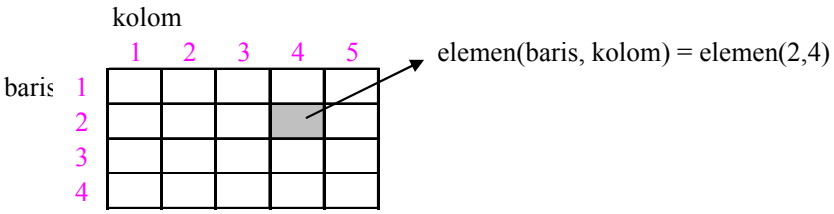
A adalah matrix koefisien dari ruas kiri persamaan
x adalah matrix kolom peubah
b adalah matrix kolom ruas kanan persamaan

sehingga dapat ditulis

$$\begin{vmatrix} 2 & -6 & 4 & 12 \\ 3 & 2 & -8 & -8 \\ 1 & 10 & 8 & -7 \\ 4 & -5 & -9 & 11 \end{vmatrix} \begin{vmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{vmatrix} = \begin{vmatrix} -10 \\ 12 \\ 8 \\ 21 \end{vmatrix}$$

14.2. DEFINISI MATRIX

Matrix adalah struktur data di dalam memori utama, dimana setiap elemennya menggunakan dua index (biasanya disebut baris dan kolom)
Karena ada dua index, disebut larik dua demensi, sedangkan istilah baris dan kolom hanyalah digunakan untuk memudahkan pengertian tentang struktur logik



Karena Matrix adalah sebuah Larik, maka konsep umum dari larik berlaku untuk Matrix

1. Kumpulan elemen bertipe sama, tipe elemen Matrix dapat berupa tipe dasar (*integer, real, boolean, char* dan *string*), atau tipe terstruktur seperti *record*
2. Setiap elemen dapat diakses secara acak (*random*) jika indexnya (baris dan kolom) diketahui, dalam hal ini index menyatakan posisi relatif didalam kumpulannya
3. Merupakan struktur data yang statis, artinya jumlah elemennya sudah ditentukan terlebih dahulu dalam Deklarasi dan tidak dapat diubah selama pelaksanaan program

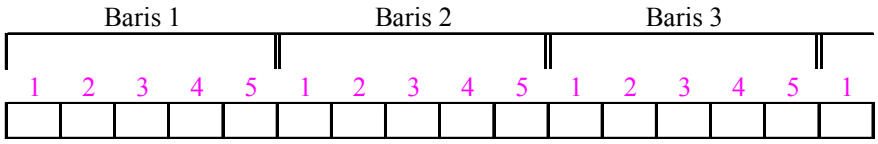
Struktur matrix praktis untuk dipakai (pengaksesan cepat, karena angsung melalui indexnya) tetapi memakan ruang memori. Misal Matrix ukuran 100 x 100, membutuhkan 10.000 x ukuran integer (2 byte)

Matrix berikut ini digambarkan dengan notasi setiap elemennya

	1	2	3	4	5
1	M[1,1]	M[1,2]	M[1,3]	M[1,4]	M[1,5]
2	M[2,1]	M[2,2]	M[2,3]	M[2,4]	M[2,5]
3	M[3,1]	M[3,2]	M[3,3]	M[3,4]	M[3,5]
4	M[4,1]	M[4,2]	M[4,3]	M[4,4]	M[4,5]

14.3. PENYIMPANAN MATRIX DALAM MEMORI

Walaupun Matrix merupakan dua demensi, tetapi kenyataan didalam memori tetap sebagai deret yang berurutan.



14.4 PENDEKLARASIAN MATRIX

Sebelum Matrix digunakan untuk menyimpan data, terlebih dahulu Matrix harus diDeklarasikan Mendeklarasikan Matrix artinya menentukan nama Matrix, tipe datanya, dan ukuran Matrix. Pendeklarasian Matrix didalam teks Algoritma ditulis dalam bagian DEKLARASI

Pendeklarasian Matrix :

1. **Sebagai variabel**

```
DEKLARASI
    M : Array [1..4, 1..5 ] of integer
```

2. **Sebagai tipe baru**

```
DEKLARASI
    type Matrix : Array [1..4, 1..5 ] of integer
    M : Matrix
```

3. **Mendefinisikan Ukuran Matrix Maximum, sebagai sebuah tetapan**

```
DEKLARASI
    const NrowMax = 20          {jumlah baris maximum}
    const NcolMax = 25          {jumlah kolom maximum}

    M : Array [1..NrowMax, 1..NcolMax ] of integer
```

14.5. PENJUMLAHAN MATRIX

Menjumlahkan dua buah Matrix A + B menghasilkan Matrix C, atau A + B = C, penjumlahan dapat dilakukan apabila ukuran matrix A dan ukuran matrix B sama, ukuran Matrix C juga sama dengan A ataupun B

Penjumlah Matrix A dan B dapat didefinisikan sebagai berikut :

C[I,J] = A[I,J] + B[I,J]

```
procedure JumlahMatrix(input A:Matrix, input B:Matrix, output C:Matrix,
                       input Nrow, Ncol : integer )
    {Menjumlahkan Matrix A dan B, yaitu A+B = C}
    {K. Awal : Matrix A dan B sudah terdefinisi elemen-elemennya}
    {K. Akhir : Elemen Matrix C berisi hasil penjumlahan elemen Matrix A dan B }
```

```
DEKLARASI
    I     : integer
    J     : integer

DESKRIPSI
    for   I ¬ 1 to Nrow do
          for   J ¬ 1 to Ncol do
                C[I,J] ¬ A[I,J] + B[I,J]
          endfor
    endfor
```

14.6. TUGAS

1 Buatlah Algoritma Perkalian Matrix

2 Tuliskan dalam bahasa Pascal Algoritma tersebut

Catatan :

A[I,J] x B[J,K] = C[I,K]

ukuran Matrix A = 4 x 5

ukuran Matrix B = 5 x 3

MATRIX A

N	A	R	O	4
T	A	M	A	2
S	U	R	A	X
B	A	Y	A	Y

MATRIX B

F	A	T	K	U
N	G	K	O	R
I	P	2	X	Y