Algoritma dan Pemrograman (4 sks)

Dosen Pengampu:

Dr. Aris Puji Widodo, MT.

Tujuan Instruksional Umum (TIU)

Tujuan umum dari pemberian mata kuliah ini adalah agar mahasiswa :

1. Mempunyai dasar pengetahuan yang kuat dan terstruktur dalam paradigma pemrograman prosedural, tanpa bergantung pada salah satu bahasa pemrograman tertentu.
2. Memberikan keterampilan menggunakan salah satu bahasa pemrograman prosedural.

Tujuan Instruksional Khusus (TIK)

Setelah mengikuti mata kuliah ini, mahasiswa dapat :

1. Memahami konsep : ekspresi, type dan jenis-jenisnya, analisa kasus, paradigma prosedural.
2. Memahami konsep paradigma prosedural: perulangan, main program, prosedur, fungsi.
3. Mengaplikasikan konsep dengan membuat program-program dalam skala kecil pada paradigma prosedural dan mengimplementasi pada salah satu bahasa prosedural (misalnya bahasa C).

Pustaka

1. Liem, Inggriani, Diktat Kuliah IF223 Algoritma dan Pemrograman, Jurusan Teknik Informatika ITB, 2003.
2. Liem, Inggriani, Diktat Kuliah IF221 Dasar Pemrograman, Jurusan Teknik Informatika ITB, 1999.
3. Liem, Inggriani, Catatan Singkat Bahasa C, Departemen Teknik Informatika ITB, 2003.
4. Liem, Inggriani, Program Kecil Bahasa C, Departemen Teknik Informatika ITB, 2003.
5. Wirth, N., Algorithm and Data Structure, Prentice Hall, 1986.

Lingkup Bahasan

1. Pengertian Dasar
2. Notasi Algoritmik
3. Type Dasar dan Bentukan
4. Harga, Ekspresi, Input dan Output
5. Sequence
6. Analisa Kasus
7. Fungsi dan Prosedur
8. Perulangan
9. Pemrosesan Sekuensial
10. Array (Tabel)
11. Searching
12. Sorting
13. Sekuensial File
14. Hubungan Berulang

Pengertian Dasar

Pengertian Dasar (1)

AKSI adalah kejadian yang terjadi pada **selang waktu terbatas** (dari T0 sampai T1) dan menghasilkan **efek neto** (membandingkan pada saat T0 dengan T1) yang terdefinisi dan direncanakan.

Contoh aksi adalah: Ibu Nia MENCUCI BAJU untuk dapat dipakai pada hari-hari berikutnya.

Pengertian Dasar (2)

Pernyataan di atas memiliki lingkup yang luas, misalnya :

- **Apakah** bajunya harus di kumpulkan terlebih dahulu dari kamar atau sudah siap di tempat pencucian ?
- **Apakah** sebelum mencuci harus membeli sabun cuci dan pewangi terlebih dahulu ?
- **Apakah** yang di maksud MENCUCI BAJU untuk dapat dipakai pada hari-hari berikutnya, adalah baju sampai dijemur ?
- **Apakah** yang di maksud MENCUCI BAJU untuk dapat dipakai pada hari-hari berikutnya, adalah baju sampai di seterika dan disimpan kembali di dalam almari ?

Pengertian Dasar (3)

Supaya MENCUCI BAJU tidak menimbulkan interprestasi yang banyak, maka perlu diberikan batasan sebagai keadaan awal (**T0**) dan keadaan akhir-nya (**T1**) yang dicapai untuk merencanakan efek neto yang diinginkan.

Untuk itu ditentukan :

ASUMSI : pada rak baju kotor selalu ada potong baju kotor yang siap untuk dilakukan pencucian, dan sabun cuci serta pewangi selalu tersedia cukup.

- **T0** : adalah baju sudah ada pada rak baju kotor, yang ditempatkan pada tempat pencucian.
- **T1** : adalah baju dijemur pada tempat penjemuran dan rak baju kotor serta ember tempat mencuci dikembalikan pada tempatnya lagi.

Pengertian Dasar (4)

Dapat ditentukan urutan kejadian (proses) Ibu Nia MENCUCI BAJU adalah :

- Ambil rak baju kotor yang berisi baju kotor
- Ambil ember tempat mencuci
- Isikan air dan sabun cuci secukupnya pada ember tempat mencuci
- Masukan baju kotor
- Cuci baju kotor
- Rendam baju dengan pewangi
- Jemur baju di tempat penjemuran
- Kembalikan rak baju kotor dan ember tempat mencuci pada tempatnya semula

Pengertian Dasar (5)

Dapat ditentukan urutan kejadian (proses) Ibu Nia MENCUCI BAJU adalah :

- Ambil rak baju kotor yang berisi baju kotor
- Ambil ember tempat mencuci
- Isikan air dan sabun cuci secukupnya pada ember tempat mencuci
- Masukan baju kotor
- ***depend on*** ketebalan baju
  - Tebal : menggunakan sikat
  - Tidak tebal : -
- Cuci baju kotor
- Rendam baju dengan pewangi
- Jemur baju di tempat penjemuran
- Kembalikan rak baju kotor dan ember tempat mencuci pada tempatnya semula

Pengertian Dasar (6)

Dapat ditentukan urutan kejadian (proses) Ibu Nia MENCUCI BAJU adalah :

- Ambil rak baju kotor yang berisi baju kotor
- Ambil ember tempat mencuci
- Isikan air dan sabun cuci secukupnya pada ember tempat mencuci
- Masukan baju kotor
- ***while*** (baju kotor belum habis) ***do***
  depend on ketebalan baju
  Tebal : menggunakan sikat
  Tidak tebal : -
  Cuci baju kotor 1 potong
  {**baju kotor habis**}
- Rendam baju dengan pewangi
- Jemur baju di tempat penjemuran
- Kembalikan rak baju kotor dan ember tempat mencuci pada tempatnya semula

Notasi Algoritmik

Notasi Algoritmik (1)

Algoritma adalah solusi detil secara prosedural dari persoalan yang direpresentasikan menggunakan **Notasi Algoritmik**, dan tidak memiliki mesin pengeksekusi.

Program adalah program komputer dalam **bahasa pemrograman** yang tersedia di dunia nyata, dan memiliki mesin pengeksekusi.

Notasi Algoritmik (2)

- **Notasi algoritmik** digunakan untuk menjembatani keragaman dan kekompleksitasan bahasa pemrograman, sehingga mahasiswa dapat melakukan "**abstraksi**".
- **Notasi algoritmik** lebih berorientasikan pada *detail design* dibandingkan *coding*, sehingga mahasiswa dapat dengan mudah untuk menterjemahkan ke salah satu bahasa pemrograman.
- **Notasi algoritmik** yang dipakai adalah menggunakan notasi algoritmik ***Dr. Inggriani Liem, Teknik Informatika ITB Bandung,*** *Revisi tanggal 21-08-2003 jam 16.03 WIB*.

Notasi Algoritmik (3)

Teks algoritma terdiri dari 3 bagian:

1. Judul (Header)
2. Kamus
3. Algoritma

Komentar dituliskan diantara tanda kurung kurawal buka (**{**) dan tutup (**}**), teks yang dituliskan diantara **{** dan **}** adalah teks dalam notasi algoritmik

Notasi Algoritmik (4)

```
JUDUL NamaProgFunProc
{Spesifikasi teks algoritmik secara umum}

KAMUS
{Bagian ini digunakan untuk mendefinisikan nama
  konstanta, nama variabel,spesifikasi fungsi,
  dan spesifikasi prosedur}

ALGORITMA
{Bagian ini, semua teks yang tidak dituliskan
  diantara tanda kurung kurawal buka dan kurung
  kurawal tutup harus dianggap sebagai notasi
  algoritmik}
```

Nama

- **Nama** digunakan sebagai identifikasi nama program, nama type, nama variabel, nama konstanta, nama fungsi, dan nama prosedur.
- Semua nama hanya didefinisikan sekali dalam program
- Sebuah nama dapat dipanggil berkali-kali dalam sederetan instruksi, misalnya nama variabel, fungsi, etc.

Judul (Header)

- **Judul (Header)** bagian dari teks algoritma untuk mendefinisikan apakah teks tersebut merupakan program, fungsi, prosedur, dan modul.
- Setelah menuliskan judul disarankan untuk menuliskan spesifikasi singkat dari teks algoritma tersebut.

Kamus

- **Kamus** bagian dari teks algoritma tempat mendefinisikan nama type, konstanta, variabel/informasi, fungsi (sekaligus spesifikasinya), dan prosedur (sekaligus spesifikasinya).
- Semua nama baru dapat dipakai setelah didefinisikan dalam kamus.
- Nama variabel belum terdefinisi harganya ketika didefinisikan. Pendefinisian nama konstanta sekaligus memberikan harganya. Pendefinisian nama fungsi dilakukan sekaligus dengan domain, range, dan spesifikasinya. Pendefinisian nama prosedur sekaligus dengan pendefinisian parameter (jika ada) dan spesifikasi (Initial State, Final State, dan proses yang dilakukan).

Contoh Pendefinisian Kamus

```
KAMUS
{Nama Type, hanya untuk type yang bukan type dasar}
   type Titik : < X : integer, Y : integer >
   {koordinat kartesian}
{Nama Kostanta, harus menyebutkan type dan nilainya}
   constant g : real = 9.8
{Nama varibel, menyebutkan type}
   IMAX : integer {indek elemen terbesar}
   Ketemu : boolean {hasil pencarian, true jika ketemu}
   T : Titik {posisi pada kartesian}
   Sum : real {jumlahan semua elemen tabel}
{Spesifikasi fungsi, menyebutkan nama fungsi, domain dan range}
   Function IntToReal(i : integer) -> real
   {Mengkonversi harga i yang bertype integer menjadi bertype real}
{Spesifikasi prosedur, menyebutkan nama, parameter, IS, FS, proses}
   Procedure Tukar (input/output : I, J : integer)
   { I.S. I dan J terdefinisi I=i dan J=j
     F.S. I=j dan J=i
     Proses : menukar isi variabel I dan J }
```

Algoritma

- **Algoritma** bagian dari teks algoritmik yang berisi instruksi atau pemanggilan aksi yang sudah didefinisikan.
- Komponen teks algoritmik dalam pemrograman prosedural dapat berbentuk:
  - Instruksi dasar : *input/output*, *assigment*
  - *Sequence*
  - Analisa kasus
  - Perulangan

Terjemahan Notasi Algoritmik ke Bahasa C

Algoritmik ke Bahasa C (1)

No.	Notasi Algoritmik	Bahasa C
1.	**ASSIGMENT** <nama> ← nilai	<nama> = nilai;

2.	**KONDISIONAL**
	if <kondisi> then
	AKSI_1
		if (kondisi) {
		AKSI_1;
		}

	if <kondisi> then
	AKSI_1
	else
	AKSI_2
		if (kondisi){
		AKSI_1;
		}
		else{
		AKSI_2;
		}

Algoritmik ke Bahasa C (2)

No	Notasi Algoritmik	Bahasa C

	depend on <nama>
	kondisi_1 : AKSI_1
	kondisi_2 : AKSI_2
	kondisi_3 : AKSI_3
	.
	.
	.
	kondisi_N : AKSI_N
		if (kondisi_1){
		AKSI_1;
		}
		else if (kondisi_2){
		AKSI_2;
		}
		else if (kondisi_3){
		AKSI_3;
		}
		.
		.
		.
		else {
		AKSI_N;
		}

Algoritmik ke Bahasa C (3)

- Untuk depend on, jika <kondisi_1> … <kondisi_N> berbentuk :

  <nama_Var> = <ekspresi konstan>, maka dapat dipakai statement switch :

```
switch (nama_Var){
    case eksp_konstan_1 : AKSI_1;break;
    case eksp_konstan_2 : AKSI_2;break;
    case eksp_konstan_3 : AKSI_3;break;
    .
    .
    .
    case eksp_konstan_N : AKSI_N;break;
}
```

Algoritmik ke Bahasa C (4)

No	Notasi Algoritmik	Bahasa C

3.	**PENGULANGAN**
	while <kondisiUlang> do
	AKSI
		while (kondisiUlang){
		AKSI;
		}

	repeat AKSI until <kondisiStop>	do{ AKSI; }while (kondisiUlang);

	iterate
	AKSI_1
	stop <kondisiStop>
	
	AKSI_2
		for(;;){
		AKSI_1;
		if (kondisiStop){
		exit;
		}
		else{
		AKSI_2;
		}
		}

Algoritmik ke Bahasa C (5)

No	Notasi Algoritmik	Bahasa C

	i traversal [Awal..Akhir]
	AKSI
		/*jika Awal <= Akhir*/
		for(i=Awal;i<=Akhir;i++){
		AKSI;
		}

		/*jika Awal >= Akhir*/
		for(i=Awal;i>=Akhir;i--){
		AKSI;
		}

Algoritmik ke Bahasa C (6)

No	Notasi Algoritmik	Bahasa C
4.	**INPUT/OUTPUT** input(nama)	scanf(format, &nama);
	read(nama)	fscanf(stream, format, &nama);
	output(nama)	printf(format, nama);
	write(nama)	fprintf(stream, format, nama);

Algoritmik ke Bahasa C (7)

No	Notasi Algoritmik	Bahasa C
5.	**Type Bentukan** type namaType : < elemen1 : type1, elemen2 : type2, elemen3 : type3, ... elemenN : typeN >	typedef struct { type1 elemen1; type2 elemen2; type3 elemen3; ... typeN elemenN; }namaType;
	type satuElemen : typeSatu	typedef typeSatu satuElemen;

Type Dasar dan Bentukan

Type

- **Type** adalah representasi data dalam komputer.
- Guna type untuk mendefinisikan objek yang akan diprogram.
- Type dibagi menjadi 2, yaitu type dasar (diasumsikan ada), dan type bentukan (type yang dibentuk dari type dasar).
- Mendefinisikan type berarti :
  - Menentukan nama type dalam kamus
  - Mendefinisikan domain harga
  - Konvensi penulisan konstanta type tersebut
  - Operator yang dapat dioperasikan terhadap type tersebut.

Type Dasar

- **Type Dasar** adalah type yang sudah tersedia oleh pemroses bahasa, sehingga pemrogram dapat memakai nama type dan semua operator yang tersedia, dan mentaati domain type tersebut .
- Macam-macam **Type Dasar** :
  - Bilangan logika/boolean
  - Bilangan bulat
  - Bilangan riil
  - karakter

Bilangan Logika/Boolean

- **Nama** : boolean
- **Domain** : [true, false]
- **Konstanta** : true false
- **Operator** :

Operator Logika

Operator	Keterangan	Hasil
and	Dan	boolean
or	Atau	boolean
Xor	Eksklusif OR	boolean
not	Negasi	boolean
EQ	Ekuivalen	boolean
nEQ	Negasi dari Ekuivalen	boolean

Bilangan Bulat (1)

- **Nama : integer**
- **Domain : Z**
- **Konstanta : -34 -1 0 3 58 999**
- **Operator :**

Operator Aritmatika

Operator	Keterangan	Hasil
*	Kali	integer
+	Tambah	integer
-	Kurang	integer
/	Bagi	real
div	Bagi	integer
mod	Sisa pembagian bulat	integer
abs	Nilai absolut	integer
^	pangkat	integer

Bilangan Bulat (2)

Operator Relasional/Perbandingan

Operator	Keterangan	Hasil
<	Kurang dari	`boolean`
≤	Kurang dari sama dengan	`boolean`
>	Lebih besar	`boolean`
≥	Lebih besar sama dengan	`boolean`
=	Sama dengan	`boolean`
≠	Tidak sama dengan	`boolean`

Bilangan Riil (1)

- **Nama** : real
- **Domain** : **R**
- **Konstanta** : -0.1 0.0 2.0 3.0 9.99
- **Operator** :

Operator Aritmatika

Operator	Keterangan	Hasil
*	Kali	real
+	Tamabh	real
-	Kurang	real
/	Bagi	real
^	pangkat	real

Bilangan Riil (2)

Operator Relasional/Perbandingan

Operator	Keterangan	Hasil
<	Kurang dari	`boolean`
≤	Kurang dari sama dengan	`boolean`
>	Lebih besar	`boolean`
≥	Lebih besar sama dengan	`boolean`
=	Sama dengan	`boolean`
≠	Tidak sama dengan	`boolean`

Karakter

- **Nama** : character
- **Domain** : himpunan yang terdefinisi oleh enumerasi, misalnya
  ['0'..'9','a'..'z','A'..'Z',RETURN, SPACE]
- **Konstanta** : ditulis diantara tanda petik
  'a' '0' 's' 'S' 'F'
- **Operator** :

Operator Perbandingan

Operator	Keterangan	Hasil
=	Sama dengan	boolean
≠	Tidak sama dengan	boolean

String,sebagai type dasar khusus

- **Nama** : string
- **Domain** : kumpulan karakter yang didefinisikan pd domain character
- **Konstanta** : ditulis diantara tanda petik
  'saya' 'a' 'adalah suatu' 'S' ' '
- **Operator** :

Operator Perbandingan

Operator	Keterangan	Hasil
=	Sama dengan	boolean
≠	Tidak sama dengan	boolean

Type Bentukan (1)

- **Type Bentukan** adalah type yang dibentuk dari type dasar atau type yang sudah tersedia.
- Contoh **Type Bentukan** adalah :
  - Type Pecahan, terdiri dari <pembilan,penyebut>
  - Type Titik, terdiri dari <absis,ordinat>
  - Type JAM, terdiri dari <jam,menit,detik>
  - Type DATE, tediri dari <tanggal,bulan,tahun>
  - … etc (disesuaikan kebutuhan)
- **Type Bentukan** dapat dibentuk menjadi type bentukan yang lainnya, misalnya dari type Titik dibuat menjadi type Garis <titikAwal,titikAkhir>,

Notasi Algoritmik Type Bentukan

```
type namaType : <
                    elemen1 : type1,
                    elemen2 : type2,
                    elemen3 : type3,
                      ...
                    elemenN : typeN
                >
```

Type Titik

Contoh : **Type Titik**

{Type Titik menyatakan absis dan ordinat integer dari sumbu kartesian}

```
type Titik : <
                    X : integer, {absis}
                    Y : integer {ordinat}
                >
```

Jika dideklarasikan variabel T sbb :

T : Titik {T adalah sebuah Titik}

Cara mengakses nilai elemen T adalah :

T.X {Nilai absis bernilai integer}

T.Y {Nilai ordinat bernilai integer}

Domain : <integer,integer>

Konstanta :

<2,4> <4,5> <67,8>

Type JAM (versi-1)

Contoh : **Type JAM** (versi-1)

{Type JAM menyatakan jam dalam notasi HH:MM:SS, dengan HH bernilai[0..23], MM bernilai[0..59], dan SS bernlai[0..59] }

```
type JAM : <
                HH : integer[0..23], {jam}
                MM : integer[0..59], {menit}
                SS : integer[0..59] {detik}
           >
```

Jika dideklarasikan variabel J sbb :

J : JAM {J adalah sebuah JAM}

Cara mengakses nilai elemen J adalah :

J.HH {Nilai jam bernilai integer [0..23]}

J.MM {Nilai menit bernilai integer [0..59]}

J.SS {Nilai detik bernilai integer [0..59]}

Domain : <integer,integer,integer>

Konstanta :

<0,0,0> <15,25,30> <6,7,48>

Type JAM (versi-2)

Contoh : **Type JAM** (versi-2)

{Type JAM menyatakan jam dalam notasi HH:MM:SS, dengan HH bernilai[0..11], MM bernilai[0..59], SS bernlai[0..59], dan ampm merupakan enumerasi (am,pm) }

```
type JAM : <
                HH : integer[0..11], {jam}
                MM : integer[0..59], {menit}
                SS : integer[0..59], {detik}
              ampm : (am,pm) {menentukan siang/malam}
            >
```

Jika dideklarasikan variabel J sbb :

J : JAM {J adalah sebuah JAM}

Cara mengakses nilai elemen J adalah :

J.HH {Nilai jam bernilai integer [0..11]}

J.MM {Nilai menit bernilai integer [0..59]}

J.SS {Nilai detik bernilai integer [0..59]}

J.ampm {nilai am atau pm}

Domain : <integer,integer,integer,(am,pm)>

Konstanta :

<0,0,0,am> <15,25,30,pm> <6,7,48,am>

Harga, Ekspresi, Input, dan Output

Harga

- **Harga (nilai)** adalah suatu besaran yang bertype.
- **Harga** dapat diperoleh dari :
  - **Isi dari nama**, yaitu nama variabel atau kostanta
  - Hasil perhitungan **ekspresi**
  - Hasil yang dikirim **FUNGSI**
  - **Konstanta** tanpa diberi nama yang dipakai langsung
- **Harga** dapat dimanipulasi :
  - Diisikan ke **NAMA** variabel yang bertype sesuai dengan harga tersebut dengan instruksi assigment
  - Diacu dari nama, untuk dipakai dalam perhitungan atau ekspresi
  - Dituliskan di piranti keluaran (layar, printer,...etc)
  - Dipakai dalam ekspresi, tergantung typenya.

Pengisian Nama Variabel

- Ada dua cara untuk mengisi nama variabel dengan harga :
  - Assigment, yaitu nama variabel / kostanta
  - Dibaca dari piranti masukan
- ASSIGMENT adalah memberikan harga pada suatu nama variabel (isinya dapat bervariasi), dengan pemberian harga ini, harga lama yang diberikan tidak berlaku, tetapi yang berlaku adalah harga paling terakhir yang diberikan.

Notasi Algoritmik ASSIGMENT

```
<nama1> ← <nama2>   {harga nama2 di salin ke nama1}
                    {harga nama1 sama dengan harga nama2}
<nama>  ← <konstanta> {harga kostanta diisikan ke nama}
<nama>  ← <ekspresi>  {hasil perhitungan ekspresi diisikan}
                      {ke nama}
```

- Bagian kiri dan kanan tanda (←) harus bertype sama.
- Bagian kiri tanda (←) (<**nama**>, <**nama1**>) harus merupakan nama variabel, tidak boleh nama type, fungsi, atau prosedur.
- Bagian kanan tanda (←) dapat berupa nama variabel, nama fungsi, nama kostanta.
- Semua nama yang dipakai dalam assigment tidak boleh nama type dan nama prosedur.

Pemberian Harga dari Piranti Masukan

- Selain assigment, pengisian harga ke suatu nama variabel dapat juga dilakukan dengan melalui piranti masukan, seperti : keyboard,mouse,scanner,...etc.
- Notasi algoritmik :

```
input(<listNama>)
```

- `<listNama>` adalah satu atau lebih nama variabel
- Nama pada `<listNama>` harus berupa nama variabel tidak boleh nama yang lainnya, seperti : nama type, nama kostanta, nama fungsi, nama prosedur

Penulisan Nama Variabel

- Suatu harga yang disimpan didalam memori komputer, harus dapat dikomunikasikan dengan dunia luar untuk diinterprestasikan oleh pemakai program.
- Harga harus dituliskan pada piranti keluaran, seperti : layar, printer, scanner,...,etc.

Notasi Penulisan Nama Variabel

```
output(<listNama>) {semua harga yang disimpan dalam setiap
                    nama yang ada pada listNama akan dituliskan
                    pada piranti keluaran sesuai urutannya}
output(<kostanta>) {harga konstanta dituliskan pada piranti
                    keluaran}
output(<ekspresi>) {hasil perhitungan ekspresi dituliskan pada
                    piranti keluaran}
output(<listNama>,<konstanta>,<ekspresi>)
                   {yang dituliskan pada piranti keluaran adalah
                    semua harga sesuai dengan urutan penulisan
                    nama,konstanta,ekspresi}
```

- <listNama> adalah satu atau lebih nama : dapat nama variabel, nama konstanta, nama fungsi beserta parameternya.
- Nama-nama pada <listNama> tidak boleh nama type atau nama prosedur.
- Nama yang ditulis harus sudah terdefinisi.

Ekspresi

- **Ekspresi** adalah suatu formulasi perhitungan yang terdiri dari **operand** dan **operator**. Ekspresi dapat berbentuk infix, prefix, posfix, tetapi yang digunakan adalah **infix**.
- Hasil perhitungan ekspresi adalah sebuah harga dengan domain yang memenuhi **type operator** yang bersangkutan.
- **Operand** harus memiliki **harga**, yang dapat berbentuk **konstanta**, **nama** (harga yang dikandung oleh nama tersebut), hasil pengiriman **fungsi**, atau merupakan **ekspresi**.
- Hasil perhitungan ekspresi (sesuai **type**-nya), selanjutnya dapat dimanipulasi, ditampilkan pada piranti keluaran, atau disimpan pada suatu nama.

Ekspresi Boolean dan Numerik (1)

```
Kamus
{Nama konstanta}
    constant benar : boolean = true
    constant PHI   : real = 3.14159
{Nama Variabel}
    Ketemu : boolean
    Flag   : boolean
    i,j    : integer
    x,y    : real
```

```
Algoritma
    {bagian ini sudah didefinisikan, maka nilai
     Ketemu adalah true, Flag adalah false, i
     adalah 4,
     j adalah 0, x adalah 0.0, dan y adalah 4.5}
```

Ekspresi Boolean dan Numerik (2)

Ekspresi Boolean	Hasil	Keterangan
`true and false`	`false`	Ekspresi boolean
`benar or false`	`true`	Ekspresi boolean
`Flag`	`false`	`Flag = false`, Ekspresi boolean
`not Ketemu`	`false`	`Ketemu = true`, Ekspresi boolean
Ekspresi Numerik	**Hasil**	**Keterangan**
`i + 7`	`11`	Ekspresi integer
`i * j`	`4`	Ekspresi integer
`x + PHI`	`3.14159`	Ekspresi riil
`i + y`	`?`	Operand tidak sejenis

Sequence

Sequence (1)

- **Sequence** adalah sederetan instruksi primitif dan/atau aksi yang akan dieksekusi oleh komputer berdasarkan urutan penulisannya.
- **Sequence** urutanya dapat dituliskan dalam satu baris, dengan cara memberikan tanda titik koma (**;**).
- **Sequence** urutanya dituliskan dengan titik koma, sebaiknya dilakukan jika sequence yang urutanya dirubah tidak memberikan pengaruh terhadap program.

Sequence (2)

- **Contoh** Dibaca dua buah nilai v (kecepatan, meter/detik) dan t (waktu, detik), yang mewakili koefisien persamaan gerak lurus beraturan. Harus dihitung dan dituliskan hasilnya, jarak yang ditempuh benda yang bergerak lurus beraturan dengan kecepatan v tersebut dalam waktu t.
- **Spesifikasi**

  Input : v (kecepatan, meter/detik), integer
  dan t (waktu, detik), integer

  Proses : menghitung S = v * t

  Output : S (jarak yg ditempuh dalam meter), integer

JARAK1

```
Program JARAK1
{Dibaca v dan t, Menghitung S = v * t, dan menuliskan
   hasilnya }
```

```
Kamus
     v : integer {kecepatan, meter/detik}
     t : integer {waktu, detik}
     S : integer {jarak,(meter) yg ditempuh dalam waktu t}
                 {dan kecepatan v pd gerak lurus beraturan}
```

```
Algoritma
     input(v,t)
     S ← v * t
     output(S)
```

JARAK2

```
Program JARAK2
{Dibaca v dan t, Menghitung S = v * t, dan menuliskan hasilnya }
Kamus
    v : integer {kecepatan, meter/detik}
    t : integer {waktu, detik}
Algoritma
    input(v,t)
    output(v*t)
```

Analisa JARAK1 dan JARAK2

1. **Input** program : dalam kehidupan sehari-hari akan sangat sulit, karena jika pengguna program salah memasukan nilai akan berakibat, harga yang disimpan akan salah. Untuk kasus ini mungkin tidak terlalu fatal, karena hanya perkalian v*t, tetapi jika v-t akan berakibat fatal.
2. **Output** program : output program sangat sulit diinterprestasikan, karena hanya berupa sebuah angka saja yang tidak jelas interprestasinya.

JARAK3

```
Program JARAK3
{Dibaca v dan t, Menghitung S = v * t, dan menuliskan hasilnya }
```

Kamus

```
    v : integer {kecepatan, meter/detik}
    t : integer {waktu, detik}
```

Algoritma

```
    output('Input nilai kecepatan =')
    input(v)
    output('Input nilai waktu =')
    input(t)
    output('Jarak yang ditempuh =',v*t)
```

Analisa JARAK3

1. Penulisan seperti ini, program akan lebih mudah dioperasikan, tetapi algoritma menjadi lebih sangat rinci.
2. Tujuan menuliskan algoritma adalah untuk menuliskan “sketsa” solusi program, jadi hanya mengandung hal yang esensial. (JARAK1 dan JARAK2)
3. Sebaiknya instruksi yang sudah sangat rinci dan tidak mengandung hal esensial dikodekan secara langsung menggunakan bahasa pemrograman pada saat implementasi. (JARAK3)
4. Teks algoritma dituliskan dengan hanya mengandung hal yang esensial, karena focusnya adalah menghasilkan “sketsa” solusi saja.

Analisa Kasus

Analisa Kasus (1)

- **Analisa kasus** adalah suatu bentuk penguraian masalah menjadi beberapa sub-masalah yang saling lepas (enumerasi semua kemungkinan **kasus**).
- Dalam bahasa pemrograman **kasus**, sering kali disebut sebagai **KONDISI** yang merupakan ekspresi bernilai boolean.
- **Notasi** analisa kasus adalah **depend on** (untuk lebih dari **2** kasus).

```
depend on {deskripsi domain}
    <kondisi_1> : <ekspresi_1>
    <kondisi_2> : <ekspresi_2>
    <kondisi_3> : <ekspresi_3>
```

Analisa Kasus (2)

- **Notasi** analisa kasus khusus untuk **1** kasus, dapat dituliskan sebagai berikut :

```
if <kondisi_1> then
    <ekspresi_1>
```

```
depend on {deskripsi domain}
   <kondisi_1>     : <ekspresi_1>
```

Analisa Kasus (3)

- **Notasi** analisa kasus khusus untuk **2** kasus, dapat dituliskan sebagai berikut :

```
if <kondisi_1> then
    <ekspresi_1>
else
    <ekspresi_2>
```

```
depend on {deskripsi domain}
   <kondisi_1>     : <ekspresi_1>
   not <kondisi_1> : <ekspresi_2>
```

Analisa Kasus (4)

- **Notasi ELSE**

```
depend on {deskripsi domain}
    <kondisi_1> : <ekspresi_1>
    <kondisi_2> : <ekspresi_2>
    else        : <ekspresi_3>
```

- **Artinya** adalah :

```
depend on {deskripsi domain}
    <kondisi_1>                              : <ekspresi_1>
    <kondisi_2>                              : <ekspresi_2>
    not <kondisi_1> and not <kondisi_2> : <ekspresi_3>
```

Ex. Analisa kasus : Maximum (1)

- Dibaca 2 buah nilai integer i, dan j, i mungkin sama dengan j. Harus dituliskan nilai yang lebih besar.
- **Spesifikasi** :
  - **Input** : i dan j integer
  - **Output** : integer i atau j
  - **Proses** : menuliskan nilai yg lebih besar, dgn asumsi bahwa i dianggap lebih besar dari j jika i>=j

Ex. Analisa kasus : Maximum (2)

```
Program MAXIJ
{diberikan i dan j, menuliskan i jika i>=j,
  dan j jika j>i}
```

Kamus

```
    i,j : integer
```

Algoritma

```
    input(i,j)
    depend on (i,j)
        i >= j : output(i)
        i < j  : output(j)
```

Ex. Analisa kasus : Ranking (1)

- Dibaca 3 bilangan integer sembarang yang tidak sama nilainya i,j, dan k, maka urutkanlah bilangan tersebut dari yang terkecil ke terbesar.
- Spesifikasi :
  - **Input** : i,j,dan k adalah integer
  - **Output** : i,j,k jika i<j dan j<k
    i,k,j jika i<k dan k<j
    j,i,k jika j<i dan i<k
    j,k,i jika j<k dan k<I
    k,i,j jika k<i dan i<j
    k,j,i jika k<j dan j<i
  - **Proses** : menuliskan nilai yang dibaca dari yg terkecil ke terbesar

Ex. Analisa kasus : Ranking (2)

```
Program RANKING1
{dibaca 3 nilai i,j,dan k,nilai ketiganya tidak sama}
{harus dituliskan dari yang terkecil ke terbesar}
```

```
Kamus
    i,j,k : integer
```

```
Algoritma
    input(i,j,k)
    depend on (i,j,k)
        i<j<k : output(i,j,k)
        i<k<j : output(i,k,j)
        j<i<k : output(j,i,k)
        j<k<i : output(j,k,i)
        k<i<j : output(k,i,j)
        k<j<i : output(k,j,i)
```

Ex. Analisa kasus : Ranking (3)

```
Program RANKING2
{dibaca 3 nilai i,j, dan k, nilai ketiganya tidak sama}
{harus dituliskan dari yang terkecil ke terbesar}
```

Kamus

```
    i,j,k : integer
```

Algoritma

```
    input(i,j,k)
    depend on (i,j)
          i<j: depend on (j,k)
                    j<k: output(i,j,k)
                    j>k: depend on (i,k)
                              k>i: output(i,k,j)
                              k<i: output(k,i,j)
          i>j: depend on (i,k)
                    i<k: output(j,i,k)
                    i>k: depend on (j,k)
                              j>k: output(j,k,i)
                              j<k: output(k,j,i)
```

Fungsi

Fungsi

- **Fungsi** adalah suatu transformasi pemetaan dari suatu nilai ke nilai yang lainnya.
- Secara algoritmik, **Fungsi** akan menerima suatu nilai yang diberikan melalui parameter formal yang bertype tertentu (jika ada) dan menghasilkan suatu nilai sesuai dengan domain yang didefinisikan pada spesifikasinya.
- Secara penulisan, **Fungsi** diberi nama dan parameter formal, serta harus didefiniskan dalam kamus.
- **Fungsi** yang didefinisikan dapat dipanggil untuk dieksekusi lewat namanya, dan dengan diberikan parameter aktualnya.

Notasi Algoritmik Fungsi

function NAMAF (<lisParameterInput>) → <typeHasil>
{spesifikasi fungsi}

Kamus lokal
{semua NAMA yang dipakai dalam algoritma fungsi}

Algoritma
{deretan instruksi input,output,analisa
kasus,perulangan}
{pengiriman nilai di akhir fungsi,harus sesuai
dengan type hasil}
→ **hasil**

Pemanggilan Fungsi

```
Program PERSOALAN
{spesifikasi : input,proses,output}
```

Kamus

```
{semua NAMA yang dipakai dalam algoritma}
    function NAMAF (<lisParameterInput>) → <typeHasil>
    {spesifikasi fungsi}
```

Algoritma

```
{deretan instruksi input,output,analisa
kasus,perulangan}
    nama ← NAMAF (<lisParameterInput>)
    output(NAMAF(<lisParameterInput>))
{nilai yang dihasilkan fungsi dapat juga dipakai
dalam ekspresi}
```

Ex. Penulisan Fungsi

```
function FX_LINEAR(x:integer) → integer
{diberikan nilai x integer, menghitung f(x)=3x+2}
```

Kamus lokal

Algoritma

```
    → (3*x+2)
```

```
function FXY(x,y:integer) → integer
{diberikan nilai x, y integer, menghitung
  f(x,y)=3x+2y-3}
```

Kamus lokal

Algoritma

```
    → (3*x+2*y-3)
```

Ex. Pemanggilan Fungsi

```
Program CONTOHFUNGSI
{Dibaca x,y integer, menghitung f(x)=3x+2
 dan f(x,y)=3x+2y-3, dan menuliskan hasil perhitungan}
```

```
Kamus lokal
    x,y : integer {data}
    FX  : integer {hasil perhitungan f(x)=3x+2}
    FXY : integer {hasil perhitungan f(x,y)=3x+2y-3}
    function FX_LINEAR(x:integer) → integer
    {diberikan nilai x integer, menghitung f(x)=3x+2}
    function FXY(x,y:integer) → integer
    {diberikan nilai x, y integer, menghitung
    f(x,y)=3x+2y-3}
```

```
Algoritma
    input(x,y)
    FX  ← FX_LINEAR(x)
    FXY ← FXY(x,y)
    output(FX,FXY)
```

Prosedur

Prosedur

- **Prosedur** adalah sederetan instruksi yang diberi nama dan menghasilkan efek neto yang terdefinisi.
- Secara algoritmik, mendefinisikan **Prosedur** berarti menentukan nama prosedur serta parameter formal (jika ada) dan mendefinisikan Initial State (**I.S.**) dan Final State (**F.S.**)
- **Prosedur** dan parameter formalnya (jika ada) harus didefiniskan dalam kamus.
- **Prosedur** tanpa parameter formal (memanfaatkan NAMA pada kamus global (harus hati-hati, untuk program yang sangat besar dan implementasinya sudah banyak file)) dan **Prosedur** dengan parameter.

Parameter Formal

- **Parameter formal** adalah nama-nama variabel (listNama) yang digunakan dalam mendefinisikan prosedur, dan membuat prosedur tersebut dapat dipanggil dengan nama yang berbeda (harus bertype sama) pada saat eksekusi.
- Parameter **input** : sebagai masukan prosedur untuk menjalankan aksi. → NAMA atau Harga
- Parameter **output** : sebagai hasil nilai yang dikeluarkan oleh prosedur (sebagai tempat menyimpan nilai hasil). → NAMA
- Parameter **input/output** : sebagai masukan dan pada akhir prosedur akan dihasilkan nilai yang baru. →NAMA

Notasi Algoritmik Prosedur

```
procedure NAMAPROC (input/output :<lisParameterFormal>)
{spesifikasi I.S., dan F.S.}
Kamus lokal
{semua NAMA yang dipakai dalam badan prosedur}
Algoritma
{deretan instruksi input,output,analisa
  kasus,perulangan}
```

Pemanggilan Prosedur

```
Program PERSOALAN
{spesifikasi : input,proses,output}
```

```
Kamus
{semua NAMA yang dipakai dalam algoritma}
    procedure NAMAPROC (input/output :<lisParameterFormal>)
    {spesifikasi I.S., dan F.S.}
```

```
Algoritma
{deretan instruksi input,output,analisa kasus,perulangan}
    NAMAPROC (<lisParameterAktual>)
{nilai yang dihasilkan tidak dapat dipakai dalam ekspresi}
```

Ex. Penulisan Prosedur

```
procedure TUKAR (input/output : j,k : integer)
{untuk menukarkan 2 buah harga integer yang
  disimpan pada nama}
{I.S. = diberikan j=J dan k=K
 F.S. = j=K, dan k=J}
```

Kamus lokal

```
    Temp : integer {variabel memorisasi}
```

Algoritma

```
    Temp <- j     {Temp=j,j=j,k=k}
    j    <- k     {Temp=j,j=k,k=k}
    k    <- Temp  {Temp=j,j=k,k=j}
```

Ex. Pemanggilan Prosedur

```
Program TUKARKAN
{dibaca dua buah harga integer j, dan k,
 menyimpannya dan mempertukarkanya}
```

Kamus

```
    x,y : integer
    procedure TUKAR (input/output : j,k : integer)
    {untuk menukarkan 2 buah harga integer yang
      disimpan pada nama}
    {I.S. = diberikan j=J dan k=K
     F.S. = j=K, dan k=J}
```

Algoritma

```
    input(x,y)
    TUKAR(x,y)
    output(x,y)
```

Perulangan

Perulangan

- **Perulangan** terdiri dari dua bagian :
  - Kondisi yang mengakibatkan perulangan suatu saat berhenti
  - Badan perulangan, yaitu aksi yang harus diulang selama kondisi perulanganya dipenuhi.

Berdasarkan Kondisi Berhenti

```
repeat
     AKSI
until <kondisiStop>
```

- Aksi akan dihentikan jika kondisiStop dipenuhi (bernilai true), dan akan dilakukan perulangan jika kodisiStop belum terpenuhi (bernilai false).

Berdasarkan Kondisi Perulangan

```
while <kondisiUlang> do
     AKSI
{kondisi berhenti dicapai pada titik program ini}
```

- Aksi akan dihentikan jika kondisiUlang tidak dipenuhi (bernilai false), dan akan dilakukan perulangan jika kodisiUlang terpenuhi (bernilai true).

Berdasarkan Dua Aksi

```
iterate
    AKSI_1
stop <kondisiStop>
    AKSI_2
{kondisi berhenti dicapai pada titik program ini}
```

- Seolah-olah merupakan gabungan **repeat** dan **while**
- Melakukan perulangan secara otomatis AKSI_1 yang pertama, kemudian dilakukan test terhadap kondisiStop :
  - AKSI_2 dijalankan dan kemudian AKSI_1 berikutnya diulang, atau
  - Perulangan dihentikan, karena AKSI_1 menghasilkan kondisi berhenti.

Berdasarkan Pencacah

```
i traversal [Awal..Akhir]
    AKSI
```

- Aksi akan dihentikan jika semua range [**Awal..Akhir**] selesai dijelajahi.

Ex. Perulangan

- Tuliskanlah sebuah teks algoritma yang membaca sebuah nilai N (integer positif lebih besar nol), dan menuliskan nilai output 1,2,3,…,N berderet ke bawah.

Ex. Perulangan Repeat

```
Program CETAKBIL1
{dibaca N >= 0, menuliskan 1,2,3,...,N berderet
  kebawah, dg repeat}
```

```
Kamus
    i,N : integer
```

```
Algoritma
    input(N)
    i ← 1
    repeat
        output(i)
        i ← i+1
    until (i>N)
```

Ex. Perulangan While

```
Program CETAKBIL2
{dibaca N >= 0, menuliskan 1,2,3,...,N berderet kebawah, dg while}
```

Kamus

```
    i,N : integer
```

Algoritma

```
    input(N)
    i ← 1
    while (i<=N) do
        output(i)
         i ← i+1
    {i>N}
```

Ex. Perulangan Iterate

```
Program CETAKBIL3
{dibaca N >= 0, menuliskan 1,2,3,...,N berderet kebawah, dg iterate}
```

```
Kamus
    i,N : integer
```

```
Algoritma
    input(N)
    i ← 1
    iterate
        output(i)
    stop (i=N)
        i ← i+1
```

Ex. Perulangan Traversal

```
Program CETAKBIL4
{dibaca N >= 0, menuliskan 1,2,3,...,N
berderet kebawah, dg traversal}
```

```
Kamus
    i,N : integer
```

```
Algoritma
    input(N)
    i traversal [1..N]
        output(i)
```

Pemrosesan Sekuensial

Skema Pemrosesan Sekuensial

- Pemrosesan elemen dilakukan satu-persatu secara runut sesuai aturan tertentu
- Persoalan pengulangan suatu proses harus tetap mengutamakan efisiensi kode program
- Terdapat dua buah skema:
  1. Skema pemrosesan dengan MARK
  2. Skema pemrosesan tanpa MARK

Komponen Pemrosesan Sekuensial

- Elemen pertama {**First_elmt**}
- Elemen siap proses {**Curr_elmt**}
- Elemen selanjutnya {**Next_elmt**}
- Tanda akhir proses {**EOP**: boolean}
  - jika dengan mark maka ada elemen fiktif
  - jika tanpa mark maka ada informasi khusus dalam elemen terakhir

Pemrosesan Elemen Tipe Dasar

- Barisan Matematika

  { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }

  - setiap elemen bertipe integer
  - elemen pertama → elemen siap proses = 1
  - elemen selanjutnya = 2 dst
  - tanda akhir proses TRUE jika elemen siap proses = 10
  - skema tanpa mark

Pemrosesan Elemen Tipe Bentukan (1)

- Daftar nilai mahasiswa

```
type TMhsNilai: < NIM:integer>0,
                  MKul:string,
                  Nilai:character['A'..'E']
                >
MhsNilai : TMhsNilai
```

Misal daftar tersebut berisi:

```
{ <'4001','Logika','A'>,
  <'4003','Logika','B'>,
  <'4004','Logika','B'>,
  <'4008','Logika','A'>,
  <'9999','',''> }
```

Pemrosesan Elemen Tipe Bentukan (2)

- setiap elemen bertipe MhsNilai
- elemen pertama → elemen siap proses = <‘4001’,’Logika’,’A’>
- elemen selanjutnya = <‘4003’,’Logika’,’B’>
- tanda akhir proses = TRUE jika elemen siap proses = <‘9999’,’’,’’>
- skema dengan mark

Skema Pemrosesan Sekuensial 1

- Skema 1 dengan Mark tanpa kasus kosong:

```
inisialisasi
first_elmt
while NOT EOP do
  proses_curr_elmt
  next_elmt
{EOP}
terminasi
```

Skema Pemrosesan Sekuensial 2

- Skema 2 dengan Mark dengan kasus kosong:

```
first_elmt
if EOP then
  proses_kasus_kosong
else
  inisialisasi
  repeat
    proses_curr_elmt
    next_elmt
  until EOP
  terminasi
```

Skema Pemrosesan Sekuensial 3

- Skema 3 tanpa Mark tak ada kasus kosong:

```
inisialisasi
first_elmt
iterate
   proses_curr_elmt
stop EOP
   next_elmt
terminasi
```

Skema Pemrosesan Sekuensial 4

- Skema 4 tanpa Mark tak ada kasus kosong:

```
inisialisasi
repeat
   proses_curr_elmt
   next_elmt
until EOP
terminasi
```

Contoh Program 1

```
Program SumBil1
{jumlah N bilangan, skema 1 dengan mark tanpa kas
us Ø}
```

Kamus

```
  N : integer = 10        {batas bilangan}
  i : integer ≥ 1 {indeks penghitungan}
  S : integer ≥ 0 {hasil penjumlahan}
```

Algoritma

```
  S ← 0                   {inisialisasi}
  i ← 1                   {first_elmt}
  while (i ≤ N) do
    S ← S + i             {proses_curr_elmt}
    i ← i + 1             {next_elmt}
  {end while: i=N+1, S=1+2+3+...+N}
  output(S)               {terminasi}
```

Contoh Program 2

```
Program SumBil2
{jumlah N bilangan, skema 2 dengan mark dengan kasus Ø}
```

Kamus

```
  N : integer          {batas bilangan}
  i : integer ≥ 1      {indeks penghitungan}
  S : integer ≥ 0      {hasil penjumlahan}
```

Algoritma

```
  i ← 1                {first_elmt}
  input(N)
  if (N < i) then
    output(0)          {proses_kasus_kosong}
  else { N ≥ 1 }
    S ← 0              {inisialisasi}
    repeat
      S ← S + i        {proses_curr_elmt}
      i ← i + 1        {next_elmt}
    until (i > N)
    {end repeat: i=N+1, S=1+2+3+...+N}
    output(S)          {terminasi}
```

Analisis Program 1 dan 2

- Kendali pengulangan adalah i
- skema dengan mark
- first_elmt → curr_elmt = 1
- next_elmt = 2 dst
- jika curr_elmt = i maka
  next_elmt = i+1
- proses: menjumlahkan i ke hasil
- EOP true jika i = N+1
- terminasi: menuliskan hasil
  penjumlahan

Contoh Program 3

```
Program SumBil3
{jumlah N bilangan, skema 3 tanpa mark tanpa kasus Ø}
```

Kamus

```
n : integer ≥ 1     {batas bilangan}
i : integer ≥ 1     {indeks penghitungan}
S : integer ≥ 0     {hasil penjumlahan}
```

Algoritma

```
input(n)
S ← 0               {inisialisasi}
i ← 1               {first_elmt}
iterate
  S ← S + i         {proses_curr_elmt}
stop (i = n)        {EOP}
  i ← i + 1         {next_elmt}
{end iterate: i=N, S=1+2+3+...+N}
output(S)           {terminasi}
```

Analisis Program 3

- Kendali pengulangan adalah i
- skema tanpa mark
- first_elmt → curr_elmt = 1
- next_elmt = 2 dst
- jika curr_elmt = i maka
  next_elmt = i+1
- proses: menjumlahkan i ke hasil
- EOP true jika i = N
- terminasi: menuliskan hasil
  penjumlahan

Tabel (*Array*)

Pengertian Tabel

- Tabel = *array* = larik = vektor
- Tipe *array* mengacu pada sekumpulan elemen setipe melalui indeks
- Elemen *array* dapat diakses jika indeks terdefi nisi
- Indeks harus punya predesesor dan suksesor, misal tipe *integer* & *character*
- Domain *array* sesuai definisi indeks
- Domain isi *array* sesuai definisi jenis *array*

Deklarasi Tabel

- Deklarasi *array* berelemen tipe dasar:

```
type TabInt [1..10] of integer
T   : TabInt
```

- Deklarasi *array* berelemen tipe bentukan:

```
type Tmhs = <NIM:integer, Nama:string>
type TabMhs [1..10] of Tmhs
M   : TabMhs
```

- Deklarasi variabel bertipe *array*:

```
TI : array [1..10] of integer
TM : array [1..10] of Tmhs
```

- Cara akses dengan indeks di kanan bawah:

T_i {jika i terdefinisi}

M_8

Pemrosesan Sekuensial pada Tabel

- Jenis persoalan:
  1. Mengisi tabel
  2. Mencetak isi tabel
  3. Mencari elemen dalam tabel
  4. Mengurutkan elemen tabel

Kamus Umum Pemrosesan Tabel

```
Kamus
   constant Nmin : integer = 1    {indeks minimum}
   constant Nmax : integer = 100 {indeks maksimum}
   type TipeElmt : ... {misal: integer}
   i : integer [Nmin..Nmax]
   T : array [Nmin..Nmax] of TipeElmt
       {tabel T didefinisikan atas indeks i
        dengan elemen bertipe TipeElmt}
   procedure inisialiasi
    { persiapan sebelum pemrosesan tabel }
   procedure proses(input X:integer)
    { pemrosesan curr_elmt }
   procedure terminasi
    { akhir proses }
```

Skema Pemrosesan Tabel

- Skema 3 dengan **iterate**

```
inisialisasi
i ← Nmin                {first_elmt}
iterate
  proses(Ti)            {proses_curr_elmt}
stop (i=Nmax)
  i ← i+1               {next_elmt}
terminasi
```

- Dipersingkat → skema 5 dengan **traversal**

```
inisialisasi
i traversal [Nmin..Nmax]
  proses(Ti)
terminasi
```

Pengisian Tabel (1)

```
Program IsiTabel1
{Mengisi tabel dengan elemen masukan dari keyboard, tra
versal sebanyak N elemen}
```

Kamus

```
  constant Nmin : integer = 1      {indeks minimum}
  constant Nmax : integer = 100    {indeks maksimum}
  T : array [Nmin..Nmax] of integer
  i : integer [Nmin..Nmax]
  N : integer [Nmin..Nmax]         {indeks efektif}
```

Algoritma

```
  repeat                           {inisialisasi}
    input(N)
  until (Nmin ≤ N ≤ Nmax)
  i traversal [Nmin..Nmax]
    input(Ti)                      {proses_curr_elmt}
  terminasi
```

Pengisian Tabel (2)

```
Program IsiTabel2
{Mengisi tabel dengan elemen masukan dari keyboard, dia
khiri 999, nilai disimpan di T_Nmin..T_N}
```

Kamus

```
  constant Nmin : integer = 1      {indeks minimum}
  constant Nmax : integer = 100    {indeks maksimum}
  T : array [Nmin..Nmax] of integer
  i : integer [Nmin..Nmax]
  x : integer                      {elemen yang dibaca}
```

Algoritma

```
  i ← Nmin                         {inisialisasi}
  input(x)                         {first_elmt}
  while (x ≠ 999) and (i ≤ Nmax)
    T_i ← x                        {proses_curr_elmt}
    i ← i+1
    input(x)                       {next_elmt}
  {end of while: x=999 or i>Nmax}
  if (i > Nmax) then               {terminasi}
    output("Tabel sudah penuh")
```

Pointer

- Pointer = penunjuk ke nama yang diacu, sehingga informasi dalam nama tersebut dapat diakses
- Berisi alamat mesin
- Memungkinkan alokasi dinamik, memori baru di**alokasi** berdasarkan kendali pemrogram jika diperlukan, jika sudah tak diperlukan maka memori di**dealokasi**
- Hati-hati dalam menggunakan

Pointer dalam Bahasa C

- Deklarasi

```
<type> *<nama>
```

- Alokasi

```
<nama> = (type*) malloc (sizeof(type))
```

- Assignment

```
*<nama> = <nilai>
```

- Dealokasi

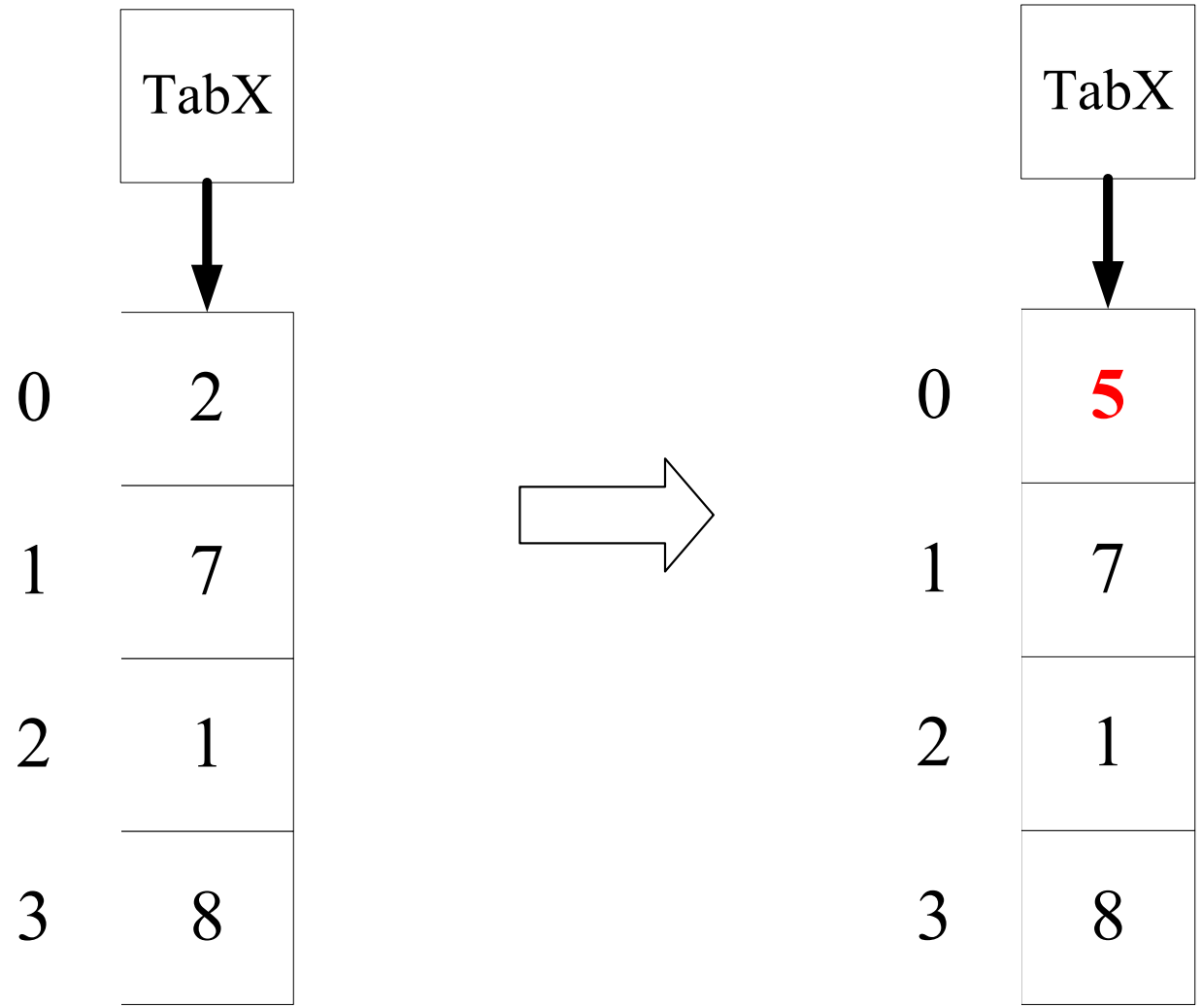
```
free ( <nama> )
```

Tabel Dinamik Bahasa C

- Dideklarasi dengan menyebut pointer ke elemen yang ke nol
- Disarankan memberi komentar untuk membedakan dengan pointer ke tipe dasar
- Contoh :

```
int *TabX; /*TabX adalah tabel integer*/
TabX = (int*) malloc (4 * sizeof(int));
*TabX = 2;       /*elemen ke-0 diisi 2*/
*(TabX+1) = 7; /*elemen ke-1 diisi 7*/
*(TabX+2) = 1; /*elemen ke-2 diisi 1*/
TabX[3] = 8;    /*elemen ke-3 diisi 8*/
TabX[0] = 5;    /*elemen ke-0 diisi 5*/
```

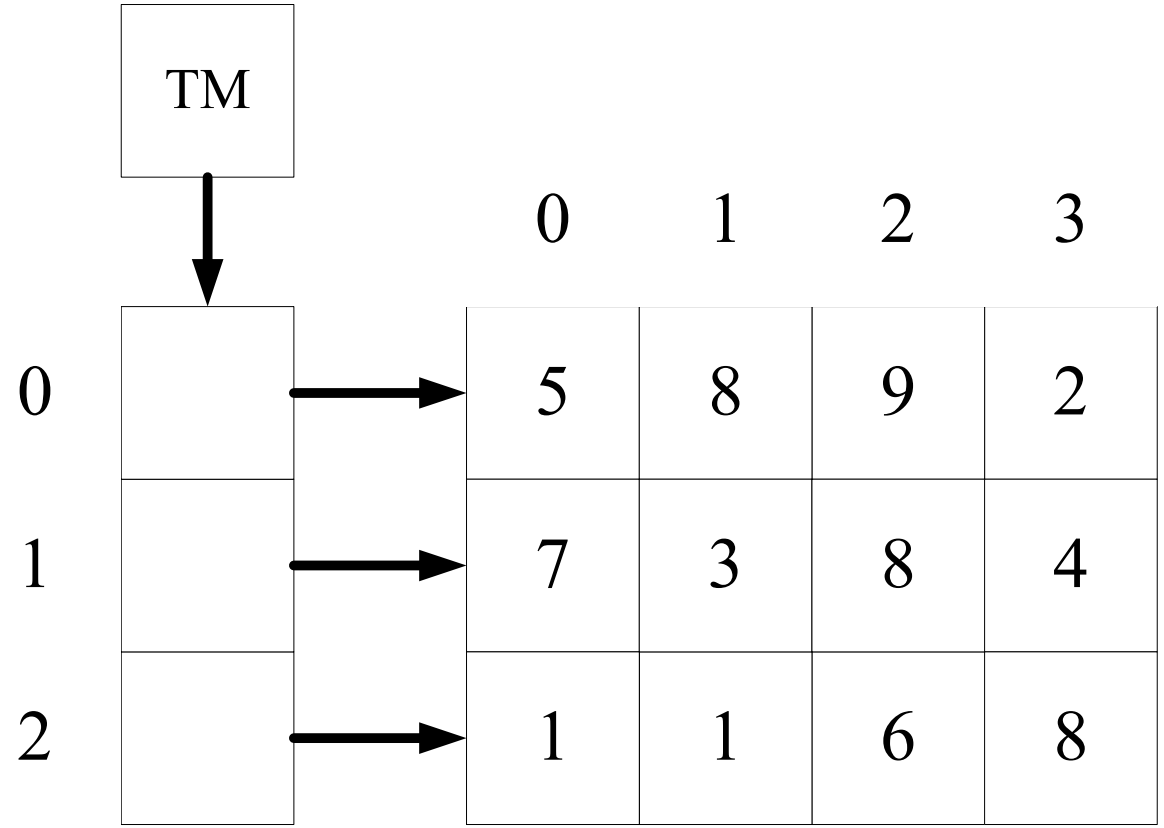
Tabel Dinamik 1 Dimensi



Tabel Dinamik Multidimensi

- Diakses dari baris lalu kolom

```
TM[1][2] = 8;
```



Pencarian (*Searching*)

Pencarian Elemen dalam Tabel

- Contoh nyata table look up: membaca daftar acara TV, jadwal kereta, melihat isi buku telepon, dll
- Kamus umum dalam pencarian

```
constant Nmax : integer = 100 {indeks maksimum}
type Tabint : array [1..Nmax] of integer
T : TabInt                    {tabel rujukan}
N : integer [1..Nmax]         {indeks efektif}
```

Algoritma Pencarian yang Dibahas

- Sequential Search
- Sorted Sequential Search
- Sequential Search with Sentinel
- Binary Search
- Maximum/Minimum Search

Sequential Search

- Pencarian satu-persatu runut dari elemen pertama sampai terakhir
- Skema 1 tanpa boolean
  - Pencarian maju, berhenti jika ketemu atau sampai di elemen terakhir
  - Elemen terakhir diperiksa khusus
- Skema 2 dengan boolean
  - Semua elemen diperiksa dalam badan pengulangan yang sama
  - Nilai tabel yang indeksnya diperiksa ada dalam definisi tabel tersebut

Sequential Search tanpa Boolean (1)

```
procedure SeqSearchX1
(input T:TabInt, N:integer, X:integer,
 output IX:integer)
{Mencari posisi terkecil X dalam T[1..N]
 hasilnya IX jika ketemu di mana T_IX = X,
 IX=0 jika tidak ketemu}
```

Kamus

```
  i : integer [1..NMax] {indeks traversal}
```

Sequential Search tanpa Boolean (2)

```
Algoritma
   i ← 1
   while (i < N) and (Ti ≠ X) do
      i ← i + 1
   {i=N or Ti=X}
   depend on T,i,X :
      Ti = X : IX ← i
      Ti ≠ X : IX ← 0
```

Sequential Search dengan Boolean (1)

```
procedure SeqSearchX2
(input T:TabInt, N:integer, X:integer,
 output IX:integer, found:boolean)
{Mencari posisi terkecil X dalam T[1..N]
 hasilnya IX jika ketemu, T_IX=X, found=true,
 IX=0 & found=false jika tidak ketemu}
```

```
Kamus
  i : integer [1..Nmax+1] {indeks traversal}
```

Sequential Search dengan Boolean (2)

```
Algoritma
   found ← false {awal search belum ketemu}
   i ← 1
   while (i ≤ N) and (not found) do
      if Ti = X then
         found ← true
      else
         i ← i + 1
   {i>N or found}
   depend on found :
      found     : IX ← i
      not found : IX ← 0
```

Sorted Sequential Search (1)

- Pencarian runut dalam tabel terurut

```
procedure SeqSearchSorted
(input T:TabInt, N:integer, X:integer,
 output IX:integer)
{Mencari posisi terkecil X dalam T[1..N]
 hasilnya IX jika ketemu, T_IX=X,
 IX=0 jika tidak ketemu}
```

```
Kamus
  i : integer [1..Nmax] {indeks traversal}
```

Sorted Sequential Search (2)

```
Algoritma
   i ← 1
   while (i < N) and (Ti < X) do
      i ← i + 1
   {i=N or Ti≥X}
   depend on T,i,X :
      Ti = X : IX ← i
      Ti ≠ X : IX ← 0 {Ti>X}
```

Sequential Search with Sentinel

- Menambahkan elemen yang dicari sebagai elemen awal/akhir tabel (sentinel), ukuran tabel bertambah
- Pencarian runut hingga ketemu atau hingga ketemu sentinel
- Kamus dengan sentinel

```
constant Nmax : integer = 100 {indeks maksimum}
type Tabint : array [1..Nmax+1] of integer
T : TabInt                    {tabel rujukan}
N : integer [1..Nmax+1] {indeks efektif}
```

SeqSearchSentinel(1)

```
procedure SeqSearchSentinel
(input T:TabInt, N:integer, X:integer,
 output IX:integer)
{Mencari posisi terkecil X dalam T[1..N]
 hasilnya IX jika ketemu, T_IX=X,
 IX=0 jika tidak ketemu}
```

Kamus

```
  i : integer [1..Nmax] {indeks traversal}
```

SeqSearchSentinel(2)

```
Algoritma
   T[N+1] ← X    {pasang sentinel di akhir}
   i ← 1
   while (T[i] ≠ X) do
      i ← i + 1
   {T[i] = X, periksa apakah sentinel}
   depend on i,N :
      i < N+1 : IX ← i {elemen tabel}
      i = N+1 : IX ← 0 {sentinel}
```

Binary Search

- Pencarian biner/dikotomik untuk tabel terurut membesar
- Bandingkan X dengan elemen tengah
- Jika sama berarti ketemu
- Jika X lebih kecil, pencarian dengan batas bawah = elemen tengah
- Jika X lebih besar, pencarian dengan batas atas = elemen tengah
- Dua versi: dengan dan tanpa boolean

BinSearch1 (1)

```
procedure BinSearch1
(input T:TabInt, N:integer, X:integer,
 output IX:integer, found:boolean)
{Mencari posisi X dalam T[1..N] dikotomik
 hasilnya IX jika ketemu, T_IX=X & found=true
 IX=0 jika tidak ketemu & found=false,
 tabel T terurut membesar T_1≤T_2≤T_3≤..≤T_N}
```

Kamus

```
  atas, bawah, tengah : integer
  {indeks batas atas, bawah, tengah}
```

BinSearch1 (2)

```
Algoritma
  atas  ← 1
  bawah ← N
  found ← false
  IX ← 0
  while (atas ≤ bawah) and not found do
    tengah ← (atas + bawah) div 2
    depend on T, tengah, X :
      X = T_tengah : found ← true
                     IX ← tengah
      X < T_tengah : bawah ← tengah - 1
      X > T_tengah : atas  ← tengah + 1
 {atas > bawah or found}
```

BinSearch2 tanpa Boolean (1)

```
procedure BinSearch2
(input T:TabInt, N:integer, X:integer,
 output IX:integer, found:boolean)
{Mencari posisi X dalam T[1..N] dikotomik
 hasilnya IX jika ketemu, T_IX=X & found=true
 IX=0 jika tidak ketemu & found=false,
 tabel T terurut naik T_1≤T_2≤T_3≤..≤T_N}
```

Kamus

```
  atas, bawah, tengah : integer
  {indeks batas atas, bawah, tengah}
```

BinSearch2 tanpa Boolean (2)

Algoritma

```
atas  ← 1; bawah ← N
tengah ← (atas + bawah) div 2
while (atas < bawah) and (X ≠ T_tengah) do
  depend on T, tengah, X :
    X < T_tengah : bawah ← tengah - 1
    X > T_tengah : atas  ← tengah + 1
  tengah ← (atas + bawah) div 2
{atas ≥ bawah or X=T_tengah}
if (X = T_tengah) then
   found ← true
   IX ← tengah
else {X ≠ T_tengah}
   found ← false
   IX ← 0
```

Maximum/Minimum Search

- Contoh nyata: mencari nilai tertinggi, mencari data percobaan minimum, dll
- Algoritma yang dibahas:
  - mencari nilai elemen ekstrim
  - mencari indeks elemen ekstrim
  - mencari maksimum dengan elemen dummy

Procedure Max1

```
procedure Max1(input T:TabInt, N:integer,
 output max:integer)
{Mencari nilai maksimum T[1..N]
 I.S.: tabel tak kosong, N>0
 F.S.: max berisi nilai maksimum}
```

Kamus

```
  i : integer {indeks traversal}
```

Algoritma

```
  max ← T1 ; i ← 2
  while i ≤ N do
    if (max < Ti) then
       max ← Ti
    i ← i + 1
  {i>N, semua elemen telah diperiksa}
```

Procedure Max2

```
procedure Max2(input T:TabInt, N:integer,
 output imax:integer)
{Mencari nilai maksimum T[1..N]
 I.S.: tabel tak kosong, N>0
 F.S.: imax = indeks elemen maksimum}
```

Kamus

```
  i : integer {indeks traversal}
```

Algoritma

```
  imax ← 1 ; i ← 2
  while i ≤ N do
    if (Timax < Ti) then
       imax ← i
    i ← i + 1
  {i>N, semua elemen telah diperiksa}
```

Procedure Max3

```
procedure Max3(input T:TabInt, N:integer,
 output max:integer)
{Mencari nilai maksimum T[1..N]
 I.S.: tabel positif, mungkin kosong, N≥0
 F.S.: max = elemen maksimum atau dummy}
```

Kamus

```
  i : integer {indeks traversal}
```

Algoritma

```
  max ← -9999 {elemen dummy}
  i ← 1
  while i ≤ N do
    if (max < Ti) then
       max ← Ti
    i ← i + 1
  {i>N, semua elemen telah diperiksa}
```

Pengurutan (*Sorting*)

Pengurutan Elemen Tabel

- Pengurutan Tabel = *Internal Sorting*
- Pengurutan data sering dilakukan dalam pengolahan data
- Pengurutan **internal**: dilakukan terhadap data yang disimpan dalam media internal (memori utama) = pengurutan tabel
- Pengurutan **eksterna**l: dilakukan terhadap data yang disimpan dalam media sekunder, biasanya berukuran besar

- Kamus umum pengurutan tabel

```
constant Nmax : integer = 100 {indeks maksimum}
type Tabint : array [1..Nmax] of integer
T : TabInt                    {tabel rujukan}
N : integer [1..Nmax]         {indeks efektif}
```

- Persoalan:
  Diberikan tabel integer T[1..N] yang isinya sudah terdefinisi. Tuliskan algoritma pengurutan elemen tabel secara membesar.
  $T_1 \leq T_2 \leq T_3 \leq \ldots \leq T_N$

Algoritma Pengurutan yang Dibahas

- Pengurutan dengan Pencacahan
- Pengurutan dengan Pemilihan
- Pengurutan dengan Penyisipan
- Pengurutan dengan Pertukaran Nilai

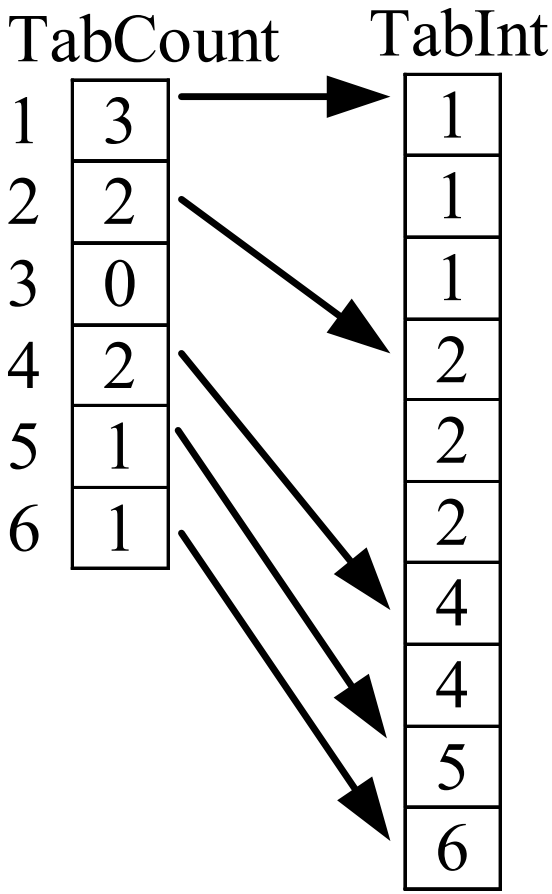
Pengurutan dengan Pencacahan

- **CountSort:**
  Diketahui data yang akan diurut memiliki daerah jelajah tertentu dan merupakan bilangan bulat misalnya [Nmin..Nmax]
- Sediakan tabel TabCount [Nmin..Nmax] yang diinisialisasi dengan 0 dan pada akhir proses $TabCount_i$ berisi banyak data pada tabel asal yang berharga i
- Bentuk kembali tabel dengan menuliskan berdasarkan TabCount

Ilustrasi CountSort

	TabCount
1	0
2	0
3	0
4	0
5	0
6	0

TabInt
1
4
2
5
1
4
6
1
2
2



Procedure CountSort(1)

```
procedure CountSort
(input/output T:TabInt, input N:integer)
{Mengurutkan tabel integer[1..N] dengan
pencacahan}
```

Kamus

```
{valmin dan valmax adalah nilai minimum dan ma
ksimum yang terdefinisi sebagai elemen T}
  TabCount : array  [valmin..valmax]
   of integer       {tabel pencacah}
  i : integer       {indeks traversal}
  K : integer       {banyak elemen}
  Ntimes : integer  {banyak pengisian elemen}
```

Procedure CountSort(2)

Algoritma

```
{Inisialisasi TabCount}
i traversal [valmin..valmax]
  TabCount_i ← 0
{pencacahan}
i traversal [1..N]
  TabCount_T_i ← TabCount_T_i + 1
{pengisian kembali}
K ← 0
i traversal [valmin..valmax]
  if (TabCount_i ≠ 0) then
    Ntimes ← K + TabCount_i
    K traversal [K+1..Ntimes]
      T_K ← I
```

Pengurutan dengan Pemilihan

- **Maxsort:**
  Mencari nilai elemen maksimum lalu menukarnya dengan elemen terujung, elemen terujung ini diisolasi dan tidak disertakan dalam proses selanjutnya
- Menghasilkan tabel terurut **mengecil**
  $T_1 \geq T_2 \geq T_3 \geq \ldots \geq T_N$

Procedure MaxSort(1)

```
procedure MaxSort
(input/output T:TabInt, input N:integer)
{Mengurutkan tabel integer[1..N] dengan
elemen mengecil dengan maksimum suksesif}
```

```
Kamus
   i : integer          {indeks traversal}
   pass : integer       {tahap pengurutan}
   temp : integer       {penampung sementara}
   imax : integer
   {indeks dimana T[1..pass] maksimum}
```

Procedure MaxSort(2)

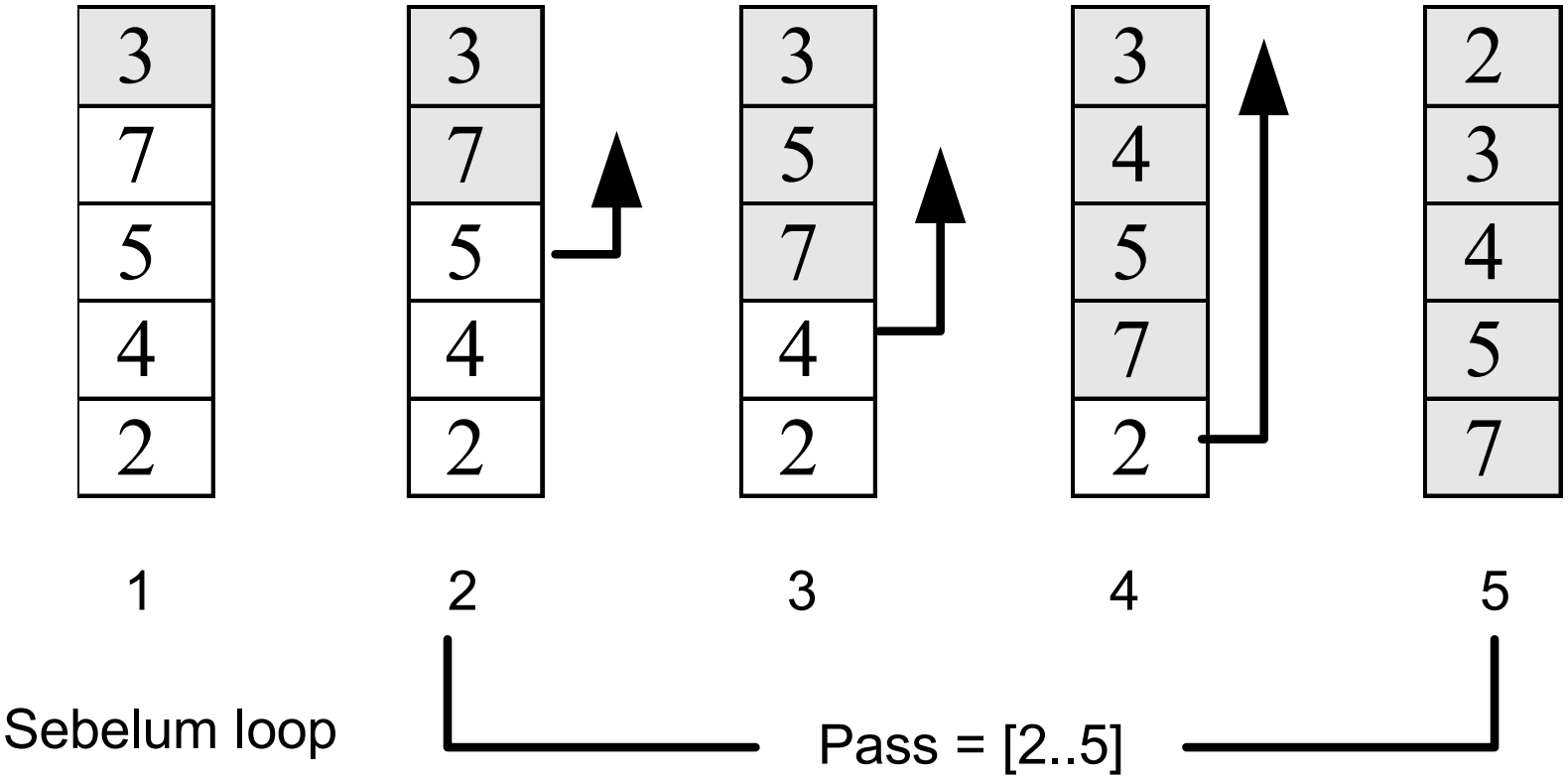
Algoritma

```
pass traversal [1..N-1]
   {tentukan maksimum [pass..N]}
   imax ← pass
   i traversal [pass+1..N]
      if T_imax < T_i then
         imax ← i
   {imax adalah maksimum T[pass..N]}
   {tukar T_imax dengan T_pass}
   temp ← T_imax
   T_imax ← T_pass
   T_pass ← temp
   {T[1..pass] terurut mengecil}
{T[1..N] terurut mengecil}
```

Pengurutan dengan Penyisipan

- **InsertionSort:**
  Mencari tempat yang tepat untuk setiap elemen tabel, dengan *sequential search*, setiap kali menyisipkan sebuah elemen tabel yang diproses ke tempat yang seharusnya
- Menghasilkan tabel terurut **membesar**
  $T_1 \leq T_2 \leq T_3 \leq \ldots \leq T_N$

Ilustrasi InsertionSort



Procedure InsertionSort(1)

```
procedure InsertionSort
(input/output T:TabInt, input N:integer)
{Mengurutkan tabel integer[1..N] dengan
menyisipkan}
```

```
Kamus
   i : integer          {indeks traversal}
   pass : integer       {tahap pengurutan}
   temp : integer       {penampung sementara}
```

Procedure InsertionSort(2)

Algoritma

```
{T1 terurut}
pass traversal [2..N]
   temp ← T_pass
   {sisipkan Tpass dalam T[1..pass-1] sambil menggeser}
   i ← pass - 1
   while (temp < T_i) and (i > 1) do
      T_i+1 ← T_i                {geser}
      i ← i - 1
   {temp ≤ Ti (tempat tepat) or i=1 sisip elemen pertama}
   depend on T, i, temp
      temp ≥ T_i : T_i+1 ← temp
      temp < T_i : T_i+1 ← T_i
                   T_i ← temp
   {T[1..pass] terurut membesar}
{Seluruh tabel terurut membesar}
```

Pengurutan dengan Pertukaran Nilai

- **BubbleSort:**
  mengapungkan elemen terkecil ke ‘atas’ melalui pertukaran
- Pada setiap pass, terdapat dua bagian yaitu bagian yang sudah terurut [1..pass-1] dan elemen ke-pass yang akan diapungkan

Procedure BubbleSort(1)

```
procedure BubbleSort
(input/output T:TabInt, input N:integer)
{Mengurutkan tabel integer[1..N] dengan
pengapungan}
```

Kamus

```
  i : integer          {indeks traversal}
  pass : integer       {tahap pengurutan}
  temp : integer       {penampung sementara}
```

Procedure BubbleSort(2)

Algoritma

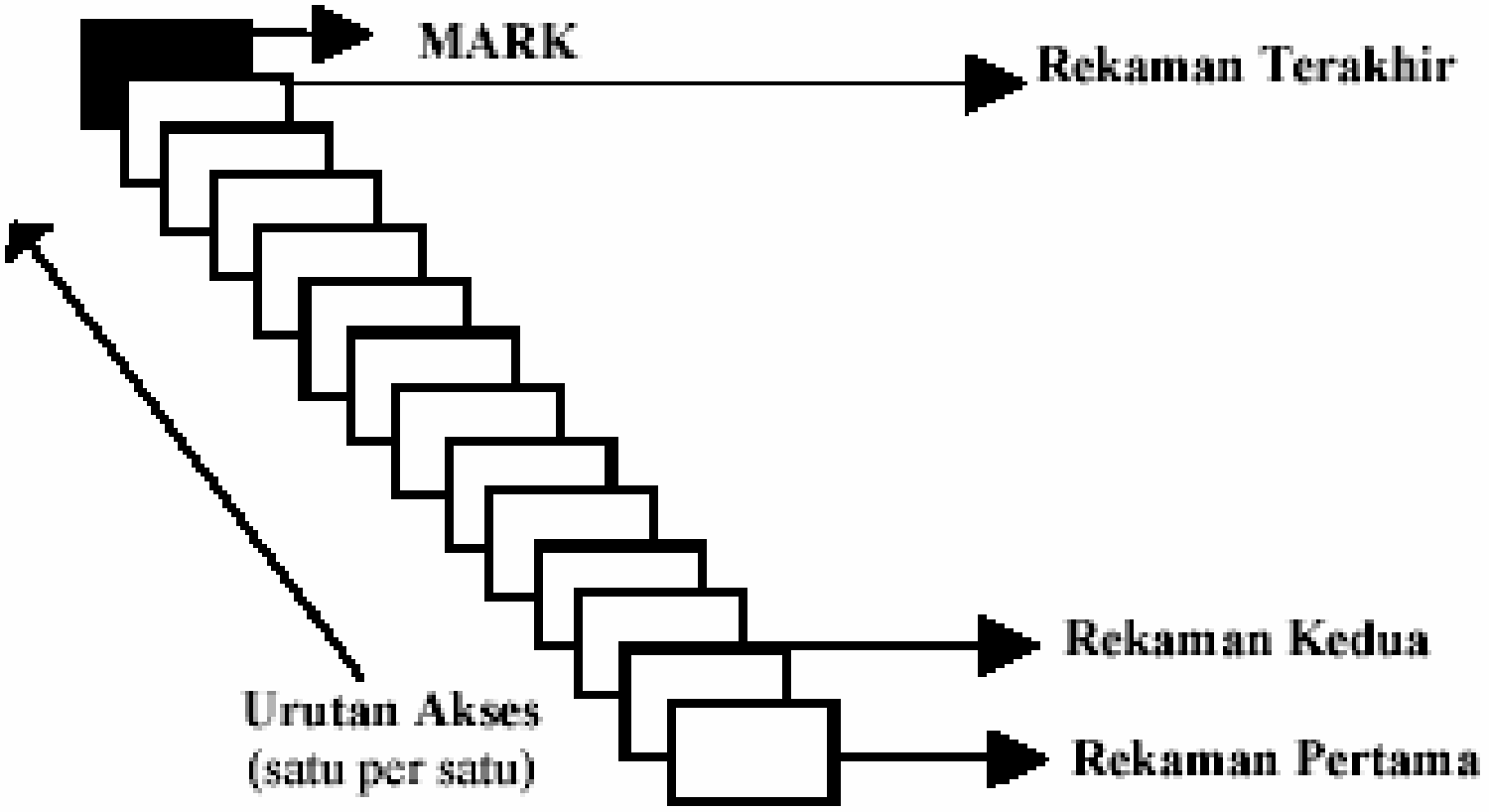
```
pass traversal [1..N-1]
  i traversal [2..N]
    if Ti < Ti-1 then
       temp ← Ti
       Ti ← Ti-1
       Ti-1 ← temp
  {T[1..pass] terurut membesar}
{Seluruh tabel terurut membesar}
```

Sequential File

Arsip Sekuensial

- Sekumpulan rekaman yang disimpan dalam media sekunder, dapat diakses secara urut dari rekaman pertama hingga terakhir secara searah
- Setiap rekaman berstruktur sama, bertipe dasar ataupun bentukan
- Elemen rekaman (field) khusus = *key*
- Jika key setiap rekaman unik maka *key* menjadi identitas rekaman = *primary key*
- Modus: dibaca saja atau ditulis saja, tidak bersamaan

Ilustrasi Arsip Sekuensial



Deklarasi Arsip Sekuensial

```
type rekaman : <…> {definisi tipe rekaman}
NamaArsip : SEQFILE of
   (*) <nama rekaman> : rekaman
   (1) <mark>
```

- (*) mungkin kosong, 1 rekaman, atau lebih

Primitif Akses Arsip Sekuensial (1)

```
procedure OPEN(input NamaArsip, <rekaman>)
{Arsip sekuensial siap dibaca
I.S.: sembarang
F.S.: informasi pada rekaman pertama siap
diakses dengan mengacu pada <rekaman>
}
```

```
procedure READ(input NamaArsip, <rekaman>)
{Rekaman setelah current_rekaman dapat dibaca
 I.S.: <rekaman> bukan mark
 F.S.: maju satu rekaman, <rekaman> berisi
 informasi rekaman setelah current_rekaman,
 mungkin <rekaman> baru = mark
}
```

Primitif Akses Arsip Sekuensial (2)

```
procedure CLOSE(input NamaArsip)
{Arsip sekuensial ditutup, tak dapat dibaca
 maupun ditulis
 I.S.: sembarang
 F.S.: arsip tak dapat diproses lagi
}
```

Primitif Perekaman Arsip Sekuensial

```
procedure REWRITE(input/output NamaArsip)
{Arsip sekuensial siap ditulis
 I.S.: sembarang
 F.S.: Arsip siap direkam di posisi pertama
}
```

```
procedure WRITE(input/output NamaArsip, <rekaman>)
{Data pada <rekaman> ditulis dalam posisi aktual,
 posisi maju satu
 I.S.: arsip berada pada posisi siap rekam,
 <rekaman> bukan mark
 F.S.: <rekaman> ditulis pada posisi aktual,
 posisi maju satu, jika <rekaman> berisi mark maka
 arsip tak dapat ditulis lagi
}
```

Contoh 1 Arsip Mahasiswa

type rekaman : <NIM:integer, nilai:integer[0..100]>
ArsipMhs : SEQFILE of
 (*) RekMhs : rekaman
 (1) <9999,0>

Konstanta: <1001,97> , <1005,86>

Akses rekaman pertama: OPEN(ArsipMhs,RekMhs)
Akses rekaman lanjut: READ(ArsipMhs,RekMhs)

Perekaman pertama: REWRITE(ArsipMhs)
Perekaman lanjut: WRITE(ArsipMhs,RekMhs)
WRITE(ArsipMhs,<1007,70>)
Perekaman akhir: WRITE(ArsipMhs,<9999,0>)

Contoh 2 Arsip Teks

type rekaman : character
Dokumen : SEQFILE of
 (*) CC : rekaman
 (1) <'#'>

Domain setiap rekaman: character
Konstanta: <'A'> , <'7'> , <'#'>

Akses rekaman pertama: OPEN(Dokumen,CC)
Akses rekaman lanjut: READ(Dokumen,CC)

Perekaman pertama: REWRITE(Dokumen)
Perekaman lanjut: WRITE(Dokumen,CC)
WRITE(Dokumen,<'A'>)
Perekaman akhir: WRITE(Dokumen,<'#'>)

Algoritma Arsip Sekuensial

- Pemrosesan sebuah arsip sekuensial
- Algoritma konsolidasi
  - Tanpa separator
  - Dengan separator
- Pemrosesan dua arsip sekuensial
  - Merging
  - Updating
  - Splitting

Pemrosesan Sebuah Arsip Sekuensial (1)

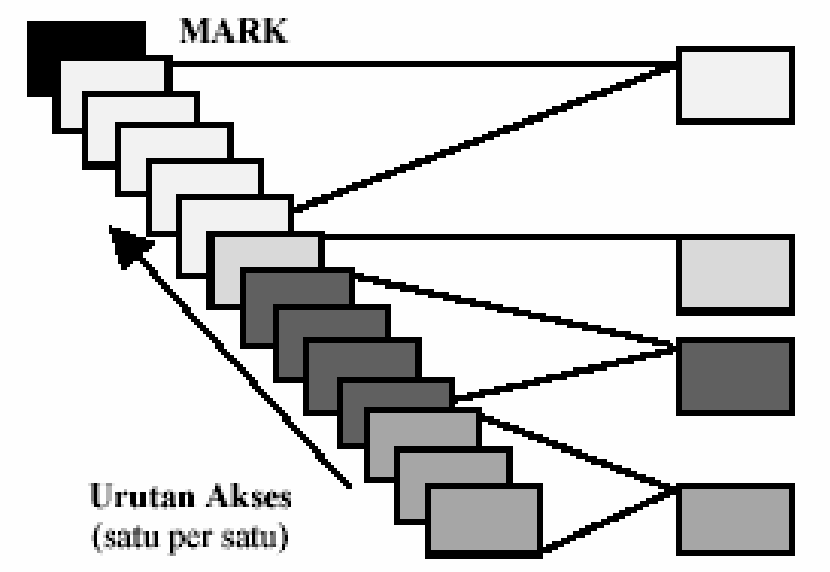
```
program NilaiRataan
{model sekuensial dengan mark, ada
  penanganan kasus kosong}
Kamus
  type rekaman : <NIM:integer,
  nilai:integer[0..100]>
  ArsipMhs : SEQFILE of
      (*) RekMhs : rekaman
      (1) <9999,0>
  SumNil : integer {jumlah nilai}
  NbMhs : integer {banyak mahasiswa}
```

Pemrosesan Sebuah Arsip Sekuensial (2)

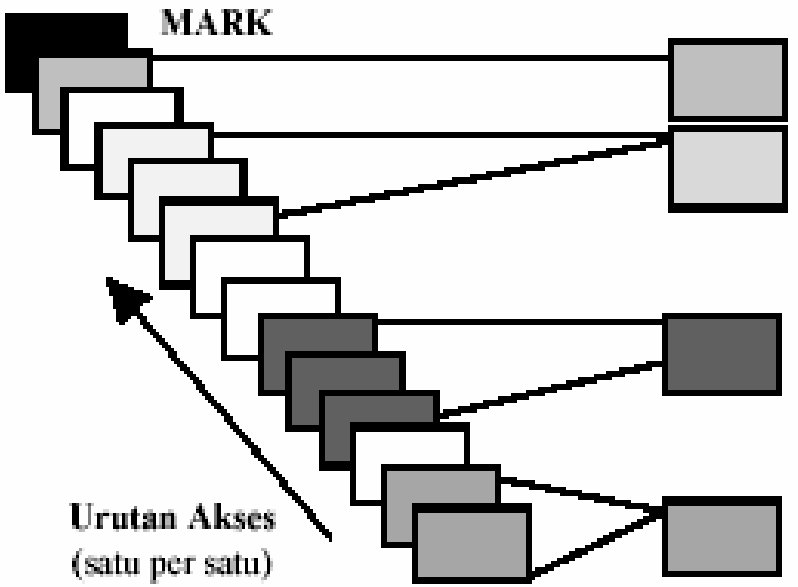
```
Algoritma
  OPEN(ArsipMhs,RekMhs)                {first_elmt}
  if RekMhs=<9999,0> then
    output('Arsip kosong')
  else
    SumNil ← 0                         {inisialisasi}
    NbMhs ← 0
    repeat
      SumNil ← SumNil + RekMhs.nilai {proses}
      NbMhs ← NbMhs + 1
      READ(ArsipMhs,RekMhs)            {next_elmt}
    until (RekMhs.NIM = 9999)          {EOP}
    output(SumNil/NbMhs)               {terminasi}
  CLOSE(ArsipMhs)
```

Algoritma Konsolidasi

- Tanpa separator



- Dengan separator



Konsolidasi Tanpa Separator (1)

```
program KonsolidasiTanpaSeparator
  {Dengan penanganan kasus kosong}
  {Input: arsip terurut}
  {Proses: mengelompokkan kategori dan
  memprosesnya}
  {Output: hasil sesuai proses}
```

Konsolidasi Tanpa Separator (2)

```
Kamus
   {KeyType adalah type kunci rekaman, terdefinisi}
   {ValType adalah type nilai rekaman, terdefinisi}
   {akhir arsip adalah mark dan Val}
   type rekaman : <KeyIn: KeyType, ValIn: ValType>
   ArsipIn : SEQFILE of    {input terurut kunci}
     (*) RekIn : rekaman
     (1) <mark, val>
   EOP : boolean           {true jika ketemu mark}
   procedure Init_All_Cat {inisialisasi global}
   procedure Term_All_Cat {terminasi global}
   procedure Kasus_Kosong {penanganan kasus kosong}
   Curr_Cat : KeyType {id kategori yang sedang diproses}
   procedure Proses_First_Elmt {inisialisasi kategori}
   procedure Init_Cat      {inisialisasi kategori}
   procedure Proses_Cur_Cat {proses elemen di kategori}
   procedure Term_Cat      {terminasi kategori}
```

Konsolidasi Tanpa Separator (3)

```
Algoritma
   OPEN(ArsipIn,RekIn) {first_elmt}
   if EOP then
     Kasus_kosong
   else
     Init_All_Cat
     repeat
       Init_Cat {inisialisasi perkategori}
       Curr_Cat ← RekIn.KeyIn
       repeat
         Proses_Cat {proses_curr_elmt}
         READ(ArsipIn,RekIn)  {next_elmt}
       until (Curr_Cat ≠ RekMhs.KeyIn)
       Term_Cat
     until EOP
     Term_All_Cat
   CLOSE(ArsipIn)
```

Konsolidasi dengan Separator (1)

```
program KonsolidasiDenganSeparator
{Dengan penanganan kasus kosong}
{Input: arsip terurut}
{Proses: mengelompokkan kategori dan
memprosesnya}
{Output: hasil sesuai proses}
```

Konsolidasi dengan Separator (2)

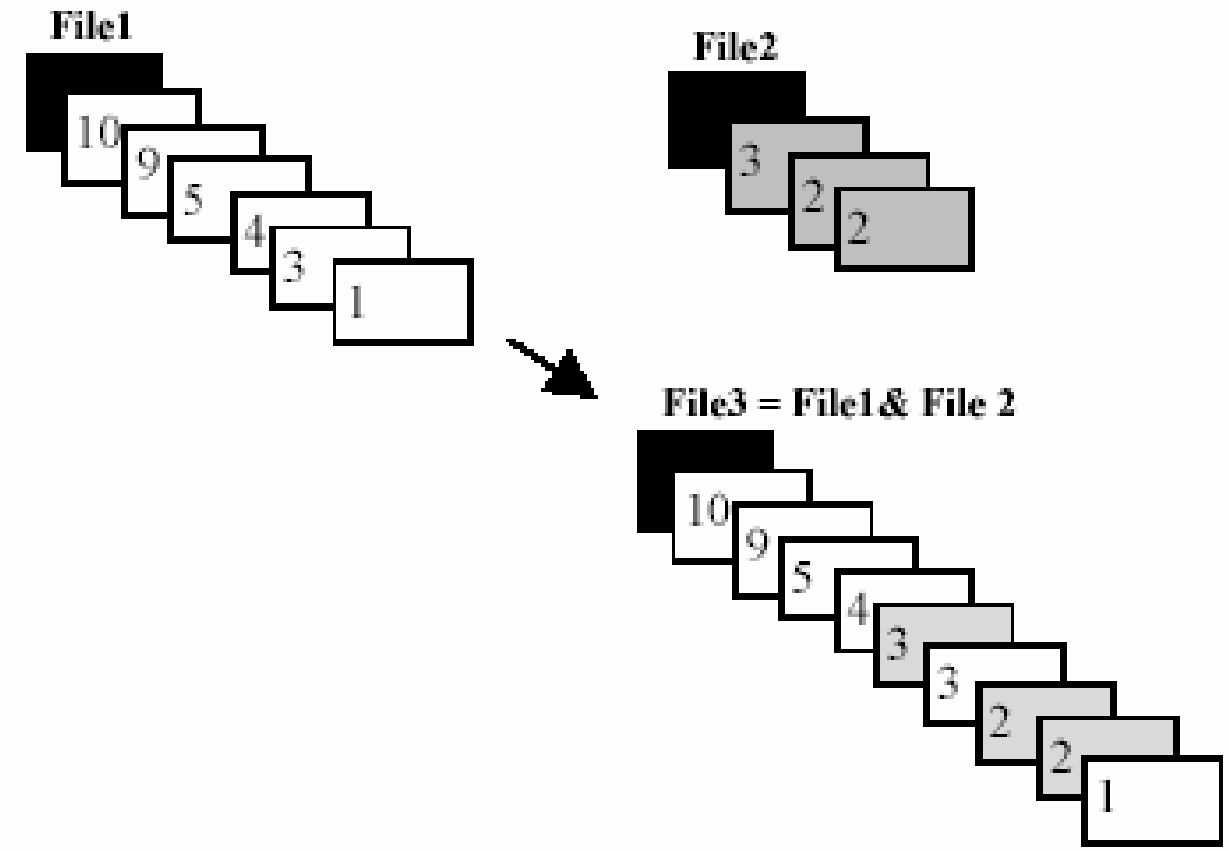
```
Kamus
  {KeyType adalah type kunci rekaman, terdefinisi}
  {ValType adalah type nilai rekaman, terdefinisi}
  {akhir arsip adalah mark dan Val}
  type rekaman : <KeyIn: KeyType, ValIn: ValType>
  ArsipIn : SEQFILE of        {input terurut kunci}
    (*) RekIn : rekaman
    (1) <mark, val>
  EOP : boolean               {true jika ketemu mark}
  procedure Init_All_Cat      {inisialisasi global}
  procedure Term_All_Cat      {terminasi global}
  procedure Kasus_Kosong      {penanganan kasus kosong}
  procedure Proses_First_Elmt {inisialisasi kategori}
  procedure Init_Cat          {inisialisasi kategori}
  procedure Proses_Cur_Cat    {proses elemen di kategori}
  procedure Term_Cat          {terminasi kategori}
  function Separator(K:KeyType)→ boolean
```

Konsolidasi dengan Separator (3)

```
Algoritma
  Init_All_Cat
  OPEN(ArsipIn,RekIn)          {first_elmt}
  if EOP then
    Kasus_kosong
  else
    repeat
      while not EOP and Separator(RekIn.KeyIn) do
        READ(ArsipIn,RekIn)    {skip separator}
      Init_Cat                 {inisialisasi perkategori}
      while not EOP and not Separator(RekIn.KeyIn) do
        Proses_Cat             {proses_curr_elmt}
        READ(ArsipIn,RekIn)    {next_elmt}
      Term_Cat
    until EOP
  Term_All_Cat
  CLOSE(ArsipIn)
```

Pemrosesan Dua Arsip Sekuensial

- Merging 2 arsip terurut mejadi 1 arsip terurut



Merging AND (1)

```
program MergingAND
   {Input: dua arsip sekuensial terurut, sejenis}
   {Proses: menggabungkan dua arsip → sebuah arsip terurut}
   {Output: arsip sekuensial terurut}
```

```
Kamus
   {KeyType adalah type kunci rekaman, terdefinisi }
   {ValType adalah type nilai rekaman, terdefinisi }
   {akhir arsip adalah mark dan Val}
   type rekaman : <Key: KeyType, Val: ValType>
   ArsipIn1 : SEQFILE of {input terurut kunci}
    (*) RekIn1 : rekaman
    (1) <mark, val>
   ArsipIn2 : SEQFILE of {input terurut kunci}
    (*) RekIn2 : rekaman
    (1) <mark, val>
   ArsipOut : SEQFILE of {input terurut kunci}
    (*) RekIOut : rekaman
    (1) <mark, val>
```

Merging AND (2)

```
Algoritma
 OPEN(ArsipIn1,RekIn1)
 OPEN(ArsipIn2,RekIn2)
 REWRITE(ArsipOut)
 while RekIn1.Key ≠ mark and RekIn2.Key ≠ mark do
  depend on RekIn1.Key, RekIn2.Key
    RekIn1.Key ≠ RekIn2.Key : WRITE(ArsipOut,RekIn1)
                              READ (ArsipIn1,RekIn1)
    RekIn1.Key > RekIn2.Key : WRITE(ArsipOut,RekIn2)
                              READ (ArsipIn2,RekIn2)
 {RekIn1.Key = mark or RekIn2.Key = mark}
```

Merging AND (3)

```
while RekIn1.Key ≠ mark do
  WRITE(ArsipOut,RekIn1)
  READ (ArsipIn1,RekIn1)
{Akhir ArsipIn1, RekIn1.Key=mark}

while RekIn2.Key ≠ mark do
  WRITE(ArsipOut,RekIn2)
  READ (ArsipIn2,RekIn2)
{Akhir ArsipIn2, RekIn2.Key=mark}

WRITE(ArsipOut,<mark,Val>)
CLOSE(ArsipIn1)
CLOSE(ArsipIn2)
CLOSE(ArsipOut)
```

Merging OR (1)

```
program MergingAND
{Input: dua arsip sekuensial terurut, sejenis}
{Proses: menggabungkan dua arsip → sebuah arsip terurut}
{Output: arsip sekuensial terurut}
```

```
Kamus
{KeyType adalah type kunci rekaman, terdefinisi }
{ValType adalah type nilai rekaman, terdefinisi }
{akhir arsip adalah mark dan Val}
type rekaman : <Key: KeyType, Val: ValType>
ArsipIn1 : SEQFILE of {input terurut kunci}
  (*) RekIn1 : rekaman
  (1) <mark, val>
ArsipIn2 : SEQFILE of {input terurut kunci}
  (*) RekIn2 : rekaman
  (1) <mark, val>
ArsipOut : SEQFILE of {input terurut kunci}
  (*) RekIOut : rekaman
  (1) <mark, val>
```

Merging OR (2)

Algoritma

```
OPEN(ArsipIn1,RekIn1)
OPEN(ArsipIn2,RekIn2)
REWRITE(ArsipOut)
while RekIn1.Key ≠ mark or RekIn2.Key ≠ mark do
  depend on RekIn1.Key, RekIn2.Key
    RekIn1.Key ≤ RekIn2.Key : WRITE(ArsipOut,RekIn1)
                              READ (ArsipIn1,RekIn1)
    RekIn1.Key > RekIn2.Key : WRITE(ArsipOut,RekIn2)
                              READ (ArsipIn2,RekIn2)
{ RekIn1.Key = mark or RekIn2.Key = mark }
WRITE(ArsipOut,<mark,Val>)
CLOSE(ArsipIn1)
CLOSE(ArsipIn2)
CLOSE(ArsipOut)
```

Updating File

- Mengubah harga rekaman yang ada pada sebuah arsip master dengan data dari arsip transaksi (*update file*)
- Arsip master terurut dengan *key* unik
- Arsip transaksi terurut, *key* tidak unik
- Satu rekaman dapat berulang kali diubah
- Jika *key* pada *update file* tidak ada di master → harus didefinisikan khusus

Updating Saldo (1)

```
program UpdatingSaldo
   {Input: dua arsip sekuensial terurut, sejenis}
   {Proses: meng-update arseip master berdasarkan transaksi}
   {Output: arsip sekuensial terurut}
   {Seq.process pada arsip master, arsip master adalah
   pengendali loop terluar}
```

```
Kamus
   type rekMaster : <KeyM: integer, Saldo: integer>
   type rekTrans  : <KeyT: integer, TransSaldo: integer>
   Master : SEQFILE of {input terurut kunci}
     (*) RekM : rekMaster
     (1) <9999,0> {mark}
   Transaksi : SEQFILE of {input terurut kunci}
     (*) RekT : rekTrans
     (1) <9999,0>
   NewMaster : SEQFILE of {input terurut kunci}
     (*) RekNM : rekMaster
     (1) <9999,0>
   NewSaldo : integer
```

Updating Saldo (2)

```
Algoritma
 OPEN(Master,RekM)
 OPEN(Transaksi,RekT)
 REWRITE(NewMaster)
 while RekM.KeyM ≠ 9999 do
  while RekT.KeyT < RekM.KeyM and RekT.KeyT ≠ 9999 do
    READ(Transaksi,RekT)
  {RekT.KeyT ≥ RekM.KeyM or RekT.KeyT = 9999}
  if RekT.KeyT = RekM.KeyM then
    NewSaldo ← RekM.Saldo
    repeat
      NewSaldo ← NewSaldo + RekT.TransSaldo
      READ(Transaksi,RekT)
    until (RekT.KeyT ≠ RekM.KeyM) or RekT.KeyT = 9999
    WRITE (NewMaster,<KeyM,NewSaldo>)
```

Updating Saldo (3)

```
  else {RekT.KeyT > RekM.KeyM, tak ada update}
    WRITE (NewMaster,<KeyM,Saldo>)
  READ(Master,RekM)
{end while RekM.KeyM = 9999}
WRITE (NewMaster,<9999,0>)
CLOSE(Master)
CLOSE(Transaksi)
CLOSE(NewMaster)
```

Splitting

- Memecah sebuah arsip menjadi dua
- Algoritma tergantung kriteria pemecahan
  - Memisah arsip pegawai sesuai golongan
  - Memisah data percobaan yang layak dan yang dibuang

Hubungan Berulang (*Reccurrence Relation*)

Hubungan Berulang (1)

- Dalam Matematika, sering ditemukan perhitungan rekursif, yang berulang dengan komponen hitung yang "mengecil"
- Contoh 1 : barisan matematika

$U_n = a + (n-1)*b$

$= U_{n-1} + b$

$S_n = U_1+U_2+ \ldots +U_n$

$= S_{n-1} + U_n$

Hubungan Berulang (2)

- Contoh 2 : faktorial

  n! = n*(n-1)*(n-2)*(n-3)* … *3*2*1

  = n*(n-1)!

  Rumus untuk fungsi faktorial F:

  F(n) : F(0) = 1

  F(n) = n * F(n-1)

Program Faktorial

```
Program Faktorial
{menghitung faktorial n!=n(n-1)(n-2)...3.2.1}
```

Kamus

```
  n : integer ≥ 0    {batas faktorial}
  i : integer ≥ 0    {indeks penghitungan}
  F : integer ≥ 1    {hasil faktorial}
```

Algoritma

```
  i ← 0
  F ← 1              {F(0) = 1}
  while i<n do
    i ← i + 1
    F ← i * F        {F(i) = i * F(i-1)}
  {end while: F=n! AND i=n }
```

Terima Kasih