Pendahuluan

Apakah Algoritma itu ?

Kata Algorism berasal dari nama penulis buku Arab yang terkenal, ***Abu Ja'far Muhammad ibnu Musa al-Kuwarizmi***, Dalam bukunya yang berjudul ***Kitab al jabr w'almuqabala***, yang artinya " ***Buku Pemugaran dan Pengurangan*** " (***The book of restoration and reduction***).

Menurut Donald. E Knuth dalam bukunya yang berjudul The Art of Computer Programming, algoritma harus mempunyai lima ciri penting :

1. Algoritma harus berhenti setelah mengerjakan sejumlah langkah-langkah yang telah ditentukan.
2. Setiap langkah yang telah ditentukan harus terdefinisi dengan jelas dan tepat sehingga tidak mempuntyai arti ganda.
3. Algoritma dapat mempunyai 0 (nol) atau lebih masukan (input)
4. Algoritma dapat mempunyai 0 (nol) atau lebih keluaran (output)
5. Algoritma harus efektif dan sederhana sehingga dapat dikerjakan dengan waktu yang masuk akal.

Dalam Kamus Besar Bahasa Indonesia terbitan Balai Pustaka 1988, dikatakan bahwa :
Algortima adalah **urutan logis pengambilan keputusan untuk pemecahan masalah.**

Algoritma sangat penting di dalam Ilmu Komputer atau Informatika, banyak cabang ilmu komputer yang dalam proses penyelesaiannya menggunakan algoritma. Namun demikian jangan beranggapan algoritma itu selalu identik dengan komputer. Karena dalam kegiatan sehari-haripun dapat kita temui istilah algoritma.

Seperti misalnya :

- algoritma untuk merakit komputer,
- Algoritma untuk menjadi mahasiswa teladan, dll.

Agar algortima dapat diproses oleh komputer, maka algoritma tersebut harus di nyatakan dalam bentuk program. Dengan kata lain bahwa program adalah implementasi dari algoritma.

Agar mudah untuk menggambarkan algoritma, maka diperlukan alat-alat untuk mengggambarkan algoritma tersebut. Namun demikian hal inipun sifatnya tidak mutlak, dapat saja si pemakai menggambarkan algoritma dengan kata-katanya sendiri selama beberapa ketentuan dan aturan khusus dapat dipenuhi.

Pemrogramman

Pemrogramman adalah merupakan langkah untuk menulis algoritma yang dapat dimengerti oleh komputer.

Langkah-langkah dalam Pemrogramman Komputer

1. Mendefinisikan Masalah

   Pada tahapan ini dicari apa masalahnya ?, apa yang harus dipecahkan oleh komputer ?, bagaimana masukan dan keluarannya ?.
2. Menentukan Solusi

   Setelah permasalahannya didefinisikan dengan jelas, masukan dan keluaran yang diminta juga jelas, maka tahap selanjutnya adalah mencari jalan bagaimana permasalahan tersebut dapat dipecahkan. Apabila permasalahan yang akan diselesaikan cukup kompleks maka penyelesaiannya dipecah ke dalam program-program yang lebih kecil yang dinamakan dengan Prosedur arau Fungsi.
3. Memilih Algoritma

   Tahap ini merupakan tahap pemilihan solusi yang telah ditemukan pada tahap sebelumnya.

   Solusi dituliskan dalam langkah-langkah penyelesaian masalah. Memilih algoritma yang baik menjadi sangat penting karena algoritma yang baik akan menghasilkan unjuk kerja program yang baik pula.
4. Menulis Program

   Setelah algoritma dipilih, selanjutnya adalah menuliskan programnya sesuai dengan bahasa pemrograman yang digunakan.
5. Menguji Program

   Setelah penulisan program selesai, selanjutnya adalah menguji program tersebut apakah sudah dapat dikompilasi dengan baik, jika sudah selanjunya adalah apakah sudah menghasilkan keluaran yang sesuai dengan kebutuhan. Untuk mendapatkan hasil yang maksimal, pengujian program sebaiknya menggunakan kasus-kasus yang banyak.
6. Menulis Dokumentasi

   Tahap ini biasanya dilakukan bersamaan dengan tahapan menulis program. Pada setiap baris program diberi komentar sedemikian rupa sehingga dapat menerangkan apa yang dilakukan oleh baris program tersebut. Hal ini diperelukan pada saat perbaikan atau perubahan program pada waktu yang akan datang.
7. Merawat Program

   Tahap ini dilakukan untuk menjaga keberlangsungan program yang sudah dibangun. Biasanya untuk mendeteksi adanya *BUG* yang tidak terdeteksi sebelumnya.

BAB I

DATA

Tes Kemampuan Awal

1. Apakah definisi algortima dan Pemrogramman ?
2. Jelaskan dan tuliskan apa yang dimaksud dengan :
   a. Tipe data dasar
      i. Bilangan Logik
      ii. Bilangan Bulat
      iii. Bilangan Riil.
      iv. Karakter
   b. Tipe data bentukan
      i. String
      ii. Record
3. Apa saja yang dapat diberi nama dalam sebuah program ?
4. Apa yang anda ketahui mengenai :
   a. Pengisian harga sebuah variabel dengan cara Penugsan,
   b. Pengisian harga sebuah variabel dengan cara Input device,
5. Apa yang anda ketahui mengenai istilah ekspresi dalam penulisan ?

DATA

Untuk mengenal lebih jauh tentang data yang sering digunakan dalam komputasi, maka ada beberapa hal yang perlu diketahui.

1.1. Tipe Data

Tipe Data dapat dikelompokan menjadi atas dua macam tipe yaitu :

- Tipe Dasar
- Tipe Bentukan.

Tipe Dasar adalah tipe yang langsung dapat dipergunakan, sedangkan tipe bentukan dibangun dari tipe dasar atau dari tipe bentukan lain yang sudah didefinisikan terlebih dahulu.

1.1.1. Tipe Data Dasar

Tipe Dasar Data terdiri dari :

- bilangan logik(boolean),
- bilangan bulat(integer),
- bilangan riil(real),
- karakter(char)

1.1.1.1. Bilangan Logik

Bilangan logik hanya mengenal 2 macam nilai yaitu :

- True (Benar)
- False (Salah)

Bilangan ini sering juga dinamakan sebagai bilangan ***Boolean***. Istilah bilangan untuk bilangan logik ini karena kondisi ***Benar*** dapat dinyatakan dengan bilangan 1 dan ***Salah*** dinyatakan dengan bilangan 0.

Tipe bilangan logik didefinisikan sbb :

Nama : boolean
Daerah : False, True
Misal : False, True
Operator : not, and, or, xor

Operasi dengan bilangan logik menghasilkan bilangan logik. Jika a dan b adalah variable yang bertipe boolean, maka hasil operasi a dan b dengan operator boolean tersebut adalah seperti dalam tabel di bawah ini :

a	b
true	false
false	true

a	b	a and b	a or b	a xor b
True	True	True	True	False
True	False	False	True	True
False	True	False	True	True
False	False	False	False	False

And hanya akan bernilai benar jika kedua-duanya benar
Or hanya akan bernilai salah jika kedua-duanya salah
Xor hanya bernilai salah jika kedua-duanya sama

Contoh :
A = True
B = True
Not A = False
Not A **and** B = False
A Or Not B = True

1.1.1.2. Bilangan Bulat

Bilangan bulat adalah bilangan yang tidak mengandung bilangan pecahan dalam penyajiannya. Misalnya : 64, 94775, - 646, 0, dsb.

Dalam algoritma, tipe bilangan bulat didefinisikan sbb :
Nama : Integer
Daerah : I
Misal : 64, - 58, 38456, 0, 3444
Operator :

a) Operator Aritmatik

+	tambah
-	kurang
*	kali
/	bagi
div	hasil bagi
mod	sisa bagi

b) Operator Relational / Perbandingan

- $<$ lebih kecil
- $\leq$ lebih kecil sama dengan
- $>$ lebih besar
- $\geq$ lebih besar sama dengan
- $=$ sama dengan
- $\neq$ tidak sama dengan

Dalam kenyataannya, bilangan bulat mempunyai daerah yang tidak terhingga ($-\infty, \infty$). Operasi aritmatika bilangan bulat dengan operator bilangan bulat menghasilkan nilai yang bertipe bilangan bulat juga.

Ada 5 tipe data yang termasuk dalam kelompok integer ini. Kelima tipe data tersebut adalah :

Tipe	Batas Nilai	Ukuran dalam byte
Byte	0..255	1
Shortint	-128..127	1
Integer	-32768..32767	2
Word	0..65535	2
Longint	-2147483648.. 2147483647	4

Contoh operasi aritmatika bilangan bulat :

13	+	10	hasil	23
86	-	16	hasil	70
2	*	8	hasil	16
48	div	3	hasil	16
48	mod	3	hasil	0
45	mod	3	hasil	0

Operasi aritmatik dengan operator relational menghasilkan nilai boolean (benar atau salah)

Contoh operasi relasional :

5	$<$	6	benar
45	$\leq$	45	benar
56	$>$	56	salah
35	$\geq$	35	benar
36	$=$	36	benar
38	$\neq$	58	benar

1.1.1.3. Bilangan Riil

Tipe bilangan riil didefinisikan sbb :

Nama : real

Daerah : R

Misal : 0. 68. 37.4 67.45 0.3456 6E-6 -4.6

Operator :

a) Operator Aritmatik

+	tambah
-	kurang
*	kali
/	bagi

b) Operator Relational / Perbandingan

$<$	lebih kecil
$\leq$	lebih kecil sama dengan
$>$	lebih besar
$\geq$	lebih besar sama dengan
$\neq$	tidak sama dengan

Semua bilangan riil harus mengandung '.' (titik).

Nilai 9 dianggap bilangan bulat, tetapi 9. dianggap bilangan riil.

Bilangan riil dapat juga dituliskan dengan notasi E yang artinya perpangkatan sepuluh.

Misalnya contoh 6E-6 artinya 6×10^{-6}.

Bilangan Bulat ataupun bilangan Riil keduanya dinamakan tipe numerik.

Ada 5 tipe data yang termasuk dalam kelompok real ini. Kelima tipe data tersebut adalah :

Tipe	Batas Nilai	Angka Signifikan	Ukuran dalam byte
Real	2.9 x 10E-39.. 1.7 x 10E-38	11 – 12	6
Single	1.5 x 10E-45.. 3.4 x 10E-38	7 – 8	4
Double	5.0 x 10E-324.. 1.7 x 10E-308	15 – 16	8
Extended	1.9 x 10E-4951.. 1.1 x 10E-4932	19 – 20	10
Comp	-2E63 + 1.. -2E63 -1	19 – 20	8

Operasi aritmatik dengan salah satu operandnya bertipe bilangan riil dinamakan operasi campuran dan menghasilkan nilai dalam daerah bilangan riil.

Contoh operasi aritmatik bilangan riil :

7.5	+	4.7	hasil	13.2
8.0	-	3.7	hasil	4.3
9	/	4	hasil	2.25
15	/	2.5	hasil	6.0

Seperti halnya pada tipe bilangan bulat, operasi relasional dengan bilangan riil menghasilkan nilai boolean.

Contoh operasi relasional :

0.067	<	34	benar
9.70	≥	5.6	benar
8.76	≠	8.76	salah

Operasi bilangan riil tidak mengenal operator " = " karena bilangan riil tidak dapat direpresentasikan secara benar oleh komputer. Misalnya 1/3 = 0.3333….. dengan angka 3 yang tidak pernah berhenti. Biasanya dalam perhitungan komputer dibulatkan sesuai dengan batasan jumlah digit dibelakang koma yang dipakai. Namun demikian, 1/3 tidaklah sama dengan 0.3333.

Dengan alasan ini maka operator " = " tidak digunakan untuk operasi bilangan riil.

1.1.1.4. Karakter

Tipe Karakter didefinisikan sbb :

Nama : char

Daerah : (0,1, ……..,9, a, b, c, …….. ,z, A,B, ……, Z, ' ', !, @, #, $, %, dan karakter khusus lainnya. (symbol ASCII)

Misal : 'h', 'Y', '1', * , dll.

Operator : relasional

- < lebih kecil
- ≤ lebih kecil sama dengan
- > lebih besar
- ≥ lebih besar sama dengan
- ≠ tidak sama dengan
- = sama dengan

Seperti pada tipe bilangan bulat, karakter mempunyai keterurutan (successor / sesudah dan predecessor / sebelum) yang ditentukan cara pengkodeannya di dalam komputer.

Contoh operasi character :

A	=	a	salah
F	=	F	benar
K	>	B	benar
Q	<	D	salah

1.1.2. Tipe Data Bentukan

Tipe bentukan dibangun dari beberapa elemen yang bertipe dasar atau dari tipe bentukan lain yang sudah didefinisikan terlebih dahulu. Operasi terhadap elemen bertipe dasar dilakukan seperti halnya pada tipe dasar. Tipe bentukan seringkali disebut juga tipe ***terstruktur*** . Tipe bentukan diberi nama oleh programmer.

1.1.2.1. String

String adalah deretan karakter dengan panjang tertentu. Tipe string didefinisikan sbb :

Nama : string

Daerah : deretan karakter yang didefinisikan pada daerah karakter

Misal : 'BANDUNG', 'UNIVERSITAS PENDIDIKAN INDONESIA',
'Jl. Soekarno - Hatta No. 211'
Semua string harus diapit oleh tanda petik tunggal.

Operator :

a) Penyambungan
+

b) Relasional

$<$	lebih kecil
$\leq$	lebih kecil sama dengan
$>$	lebih besar
$\geq$	lebih besar sama dengan
$\neq$	tidak sama dengan
=	sama dengan

Operator '+' disini bukanlah operator penjumlahan seperti pada tipe numerik (integer atau real). Operator '+' berarti penyambungan. Bila a dan b adalah peubah bertipe string, maka a + b sama dengan ab.

Contoh :

'Teknik'+'INFORMATIKA' = 'TeknikINFORMATIKA'

'Teknik'+'_INFORMATIKA' = 'Teknik_INFORMATIKA'

'xxx' + '*_yyy'* + *'zzz'* = *'xxx*_yyyzzz'

'1' + '2' = '12'

Operator relasional, seperti pada karakter, menghasilkan nilai boolean (benar atau salah).

Contoh :

'Efgh'	=	'efg'	salah
'saya'	<	'SAYA'	salah

String yang terdiri dari angka dan huruf sering dinamakan dengan *alfanumerik* .

1.1.2.2. Record / Rekaman

Sebuah record / rekaman disusun oleh satu atau lebih field. Tiap field berisi data dari tipe dasar tertentu atau tipe bentukan lain yang sudah didefinisikan terlebih dahulu. Nama record/rekaman ditentukan oleh programmer.

Sebuah rekaman / record dengan tiga buah field jika digambarkan secara *logic* adalah sbb :

Field1	Field2	Field3

Contoh :

Mendefinisikan tipe bentukan yang menyatakan data mahasiswa. Data mahasiswa terdiri atas NIM, Nama dan Usia.

Misalkan tipe bentukan ini dinamakan MHS.

Type MHS : record [NIM : integer, nama : string, usia : integer]

Jika dideklarasikan M adalah variable bertipe MHS, maka cara mengacu tiap field pada record / rekaman M adalah :

M.Nim

M.Nama

M.usia

Nama : MHS

Daerah : sesuai dengan daerah masing-masing field

Misal : [743658385, ‘Uci Sanusi’, 79]

[649460485, ‘Unang hermansyah, 34]

Operator : Tidak ada operator untuk MHS tetapi kita dapat melakukan :

- operasi integer terhadap MHS.NIM
- operasi string terhadap MHS.nama
- operasi real terhadap MHS.usia

1.2. Nama

Di dalam program, sesuatu yang diberi nama dapat berupa :

1. Variable (Peubah)
2. Constant (Tetapan)
3. Tipe seperti yang telah dijelaskan
4. Fungsi yang digunakan (dijelaskan kemudian)
5. Prosedur yang digunakan (dijelaskan kemudian)

Dalam penulisan program, ada beberapa aturan penulisan nama yang harus dipenuhi.

1. Harus selalu diawali dengan huruf (alfabet)
2. Huruf besar atau huruf kecil tidak dibedakan
3. Karakter penyusun nama hanya terdiri dari alfabet, angka, garis bawah (‘ _ ‘).

4. Tidak boleh ada spasi yang memisahkan string yang disusun.
5. Panjang nama tidak dibatasi.

Contoh yang salah :

7YANG	karena dimulai dengan angka
nilai UAS	karena ada spasi
NL-1	karena ada operator kurang
@Satu	karena ada karakter khusus
B 2	karena ada spasi

Contoh yang benar

YANG_7
nilaiUAS
NL1
Satu
B2

1.3. Harga

Harga adalah nilai besaran dari tipe yang sudah dikenal. Harga dapat berupa nilai yang dikandung oleh nama variable atau **Constant** (tetapan).

1.3.1. Pengisian Harga ke dalam Variable

Harga dapat diisikan ke dalam variable dengan cara Penugasan / *assigment*

1.3.1.1. Penugasan

Penugasan adalah mengisikan sebuah harga pada variable secara langsung. Proses penugasan biasanya dalam algoritma dapat digambarkan dengan lambang '←' atau ' = '.
Harga yang dimasukan ke dalam variable dapat juga di ambil dari variable yang lain selama tipe data yang dimasukan sama dengan tipe nama yang akan dimasukinya.
Contoh Notasi Algoritma untuk penugasan :

Nama ←	tetapan (harga tetapan diisikan ke dalam nama)
Nama1 ←	nama2 (harga nama2 disalin ke nama1, nama1 sama dengan nama2)
Nama ←	ekspresi (hasil perhitungan diisikan ke dalam nama)

Atau

Nama =	tetapan (harga tetapan diisikan ke dalam nama)
Nama1 =	nama2 (harga nama2 disalin ke nama1, nama 1 sama dengan nama2)
Nama =	ekspresi (hasil perhitungan diisikan ke dalam nama)

1.3.1.2. Input Harga dari Input device

Harga untuk Nama variable dapat diisi dari peralatan input misalnya keyboard.
Mengisi harga dari input device dinamakan operasi pembacaan data. Istilah baca ini timbul karena komputer seakan-akan membaca harga yang diberikan oleh pemakai.
Dalam Algoritma perintah untuk pembacaan data dilakukan dengan perintah **input**.
Contoh :

```
Input ( nilai1)
```

Syaratnya adalah nilai1 adalah nama variable, tidak boleh nama tetapan / constant.
Ketika perintah di atas dilaksanakan, komputer menunggu pemakai mengetikan harga nila1 dari input device.
Bila diketikan angka 96 lalu diikuti dengan tombol <enter>, maka tempat dimemori yang bernama nilai1 sekarang berisi 96.

1.3.2. Ekspresi

Suatu harga dipakai untuk proses transformasi menjadi keluaran yang diinginkan.
Transformasi harga menjadi keluaran dilakukan melalui suatu perhitungan (komputasi). Cara perhitungan itu dinyatakan dengan suatu ekspresi.
Suatu ekspresi terdiri dari ***operand*** dan ***operator Operand*** adalah harga yang dioperasikan dengan ***operator*** tertentu. **O*perand*** dapat berupa nama variable atau constant / tetapan.
Hasil perhitungan adalah harga dengan daerah yang sesuai dengan tipe ***operator*** yang dipakai.
Ekspresi dikenal dalam dua macam : Ekpresi Numerik dan Ekspresi Bolean.

1.3.2.1. Ekspresi Numerik

Ekspresi Numerik adalah ekspresi yang baik *operand*- nya bertipe numerik dan hasilnya juga bertipe numerik.
Misalkan diinisialisasikan / kamus sbb :

```
Kamus
    a, b, c  : real
    d        : integer
    e, f, g  : integer
```

Contoh Ekspresi Numerik :

```
a       x       b
```

Hasil perhitungan juga bertipe real. Pada ekspresi ini, *operand*-nya adalah a dan b sedangkan operatornya adalah “ x ” (perkalian). Bila hasil perhitungan disimpan ke dalam

nama variable, maka nama variable haruslah bertipe sama dengan tipe hasil. Pengisian hasil ekpresi a x b ke dalam variable c melalui penugasan :

c ← a x b

atau

c = a x b

1.3.2.2. Ekspresi Relasional

Ekpresi relational adalah ekspresi dengan operator $<$, $>$, $\leq$, $\geq$, $=$ dan $\neq$.Hasil ekpresinya selalu bernilai boolean.

Inisialisasi / identifikasi kamusnya adalah :

Kamus

Ada, Ketemu, besar	: boolean
x, y	: integer

Contoh :

Not ada	hasil false
ada and true	hasil true
x < 5	hasil false
ada or (x = y)	hasil true

1.3.3. Output Harga

Harga yang disimpan di dalam memori dapat ditampilkan ke output device. Perintah penulisan harga adalah dengan perintah **output**.

Contoh :

Output (nama1)
Output (constant)
Output (ekpresi)

BAB II

STRUKTUR ALGORITMA DAN ALAT PENGGAMBARAN NYA

Tes Kemampuan Awal

1. Apa yang anda ketahui mengenai struktur algoritma :
   a. Runtunan / Sequence
   b. Pemilihan / Selection
   c. Pengulangan / Looping
2. Apa yang anda ketahui mengenai alat penggambaran algoritma :
   a. Sistem Flowchart,
   b. Program Flowchart.
   c. Nassi Schneiderman,
   d. Pseudocode
3. Apa yang anda ketahui mengenai bagian algoritma :
   a. Bagian Kepala Algoritma,
   b. Bagian Deklarasi,
   c. Bagian Deskripsi.
4. Buatlah Algoritma dengan menggunakan Flowchart, Ns Diagram dan Pseudocode untuk menampilkan deret angka seperti dibawah ini :
   a. 1, 2, 3, 4, 5, …..,n
   b. 1, 3, 5, 7, 9, …..,n
   c. 2, 4, 6, 8, 10, …,n
   d. 1, 5, 9, 13, …….,n
5. Buatlah Algoritma dengan menggunakan Flowchart, Ns Diagram dan Pseudocode untuk mencari jumlah masing-masing deret yang ada pada Soal No. 1.

2.1. Struktur Algoritma

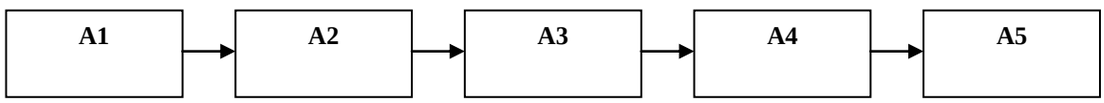
Algoritma adalah rangkaian langkah-langkah yang logis untuk menyelesaikan masalah.
Ada tiga struktur dasar untuk membangun algoritma. Ketiga struktur tersebut adalah :

2.1.1. Runtunan / Sequence

Algoritma dalah merupakan runtunan proses, yang berarti :

1. Tiap proses dikerjakan satu per satu.
2. Tiap proses dilaksanakan tepat satu kali, tidak ada proses yan diulang.
3. Urutan proses yang dilaksanakan dalam pemrosesan sama dengan urutan proses sebagaimana yang tertulis di dalam algoritma.
4. Akhir dari proses terakhir merupakan akhir algoritma.

Bila runtunan proses dalam algoritma dilambangkan dengan A1, A2, A3, A4 dan A5, maka urutan pelaksanaan proses-proses tersebut adalah seperti gambar di bawah ini :



Contoh :

Algoritma membuat Kue Bolu.
{ Catatan : semua bahan sudah tersedia dan sudah sesuai dengan takarannya }

- *Masukan Putih Telur*
- *Kocok Sampai mengembang*
- *Masukan gula pasir*
- *Kocok Sampai merata*
- *Masukan tepung*
- *Kocok Sampai merata*

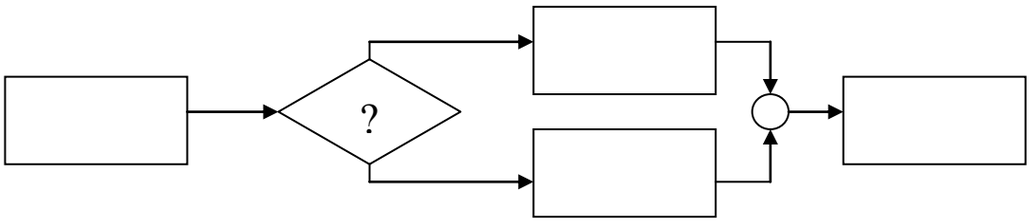
Algoritma Penulisan STMIK-MI;
Kamus Lokal :
    Baca : String [20]

Algoritma :
    Input (Baca)
    Output (Baca)

2.1.2. Pemilihan / Selection

Dalam beberapa kasus tertentu kita dihadapkan kepada persoalan yang mempunyai pilihan atau dua jawaban yang berbeda dan salah satunya harus dipilih atau bahkan mungkin mempunyai pilihan lebih dari dua.

Untuk hal yang demikian, maka pelaksanaan proses adalah seperti di bawah ini :



2.1.2.1. Notasi analisis satu masalah

Algoritma :

If kondisi **THEN**

Aksi

Endif

Aksi sesudah *then* akan dilaksanakan jika kondisi bernilai benar. Jika kondisi bernilai salah maka aksi tersebut tidak akan dikerjakan (dilewat).

Contoh:

- ***Jika*** *Amir menjadi juara kelas*

  Maka *akan dibelikan Sepeda Mini*

Contoh :

Algoritma Penulisan STMIK-MI;
Kamus Lokal :
Baca : String [20]
Agoritma :
Input (Baca)
IF Baca <> : “STMIK-MI” **THEN**
Baca =: (“Tulisan Anda Salah “)
ENDIF
Output (Baca)

2.1.2.2. Notasi analisis dua masalah

Algoritma :

IF kondisi **THEN**

Aksi1

ELSE

Aksi2

ENDIF

Aksi1 sesudah then akan dilaksanakan jika kondisi bernilai benar, jika kondisi bernilai salah maka aksi2 yang akan dikerjakan.

Contoh:

- ***Jika*** *Amir menjadi juara kelas*

  Maka *akan dibelikan Sepeda Mini*

  Selain itu *tidak mendapat apa-apa*

Contoh :

Algoritma Penulisan UNIVERSITAS A;

Kamus Lokal :

Baca : String [20]

Algoritma :

Input (Baca)

IF Baca <> : “UNIVERSITAS A” **THEN**

Baca := (“Tulisan Anda Salah “)

ELSE

Baca := (“Tulisan Anda Benar “)

ENDIF

Output (Baca)

2.1.3. Pengulangan / Looping

Selain dapat melakukan proses pemilihan aksi, komputer juga dapat melakukan aksi pengulangan.

Aksi pengulangan ini dilakukan sebanyak proses yang diinginkan atau sampai kondisi tertentu dipenuhi.

Secara umum pengulangan terdiri atas dua bagian :

1. ***Kondisi***, yaitu ekspresi boolean yang mengakibatkan pengulangan sampai suatu saat berhenti.
2. ***Aksi*** yang diulang selama belum memenuhi kondisi berhenti.

Di dalam program terdapat beberapa notasi pengulangan yang berbeda, demikian juga dengan notasi dalam algoritma. Beberapa notasi dapat digunakan untuk satu masalah yang sama, namun ada notasi pengulangan tertentu yang hanya cocok dipakai untuk permasalahan tertentu.

Beberapa macam perintah pengulangan :

2.1.3.1. Notasi FOR – Do

Notasi Algoritma :

```
For variable ← harga awal To harga akhir
        aksi
Endfor
```

Aksi di dalam badan pengulangan akan dilakukan sampai dengan ***harga akhir*** dipenuhi. Apabila satu aksi pertama telah dilakukan maka secara otomatis variable yang tadinya mempunyai harga awal akan bertambah 1.

Contoh :

```
Algoritma Pengulangan_For_do;
Kamus Lokal :
    N : Integer

Algoritma :
    Input ( N )
    FOR J ←1 TO N DO
        Output (UNIVERSITAS A)
    ENDFOR
```

Algoritma ini akan mencetak tulisan ‘UNIVERSITAS A’ sebanyak N kali.

2.1.3.2. Notasi Repeat – Until

Notasi Algoritma :

```
Repeat
        aksi
Until kondisi
```

Aksi di dalam badan pengulangan akan dilakukan selama kondisi bernilai benar. Apabila bernilai salah, maka pengulangan akan berhenti. Namun demikian diperlukan suatu perintah untuk mengubah kondisi sehingga memungkinkan untuk kondisi tidak dipenuhi. Apabila tidak ada perintah untuk mengubah kondisi yang telah ada, maka proses pengulangan tidak akan pernah berhenti.

Contoh :

Algoritma Pengulangan_Repeat_Until

Kamus Lokal :

J : Integer

Algoritma :

J ←1

REPEAT

Output (UNIVERSITAS A)

J ← J+1

UNTIL (J > 20)

Algoritma ini akan mencetak tulisan ‘UNIVERSITAS A’ sebanyak 20 kali.

2.1.3.3. Notasi While – Do

Notasi Algoritma :

While kondisi do

aksi

Endwhile

Aksi di dalam badan pengulangan akan dilakukan selama kondisi bernilai benar. Apabila bernilai salah, maka pengulangan akan berhenti. Namun demikian diperlukan suatu perintah untuk mengubah kondisi sehingga memungkinkan untuk kondisi tidak dipenuhi.

Apabila tidak ada perintah untuk mengubah kondisi yang telah ada, maka proses pengulangan tidak akan pernah berhenti.

Perbedaannya dengan ***Repeat-Until*** adalah ada kemungkinan aksi tidak dijalankan karena kondisi bernilai salah. Jika pada ***Repeat-Until*** minimal akan dilakukan aksi sebanyak 1 kali.

Contoh :

Algoritma Pengulangan_While_Do

Kamus Lokal :

J,N : Integer

Algoritma :

Input(N)

J ←1

WHILE J ≤ N **DO**

Output (UNIVERSITAS A)

J ← J+1

ENDWHILE

Setelah melakukan proses Input N maka algoritma ini akan mencetak tulisan 'UNIVERSITAS A' selama nilai J lebih kecil atau sama dengan nilai N.

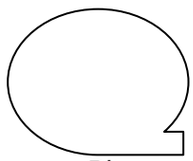
2.2. Flowchart

Tujuan utama dari penggunaan ***flowchart*** adalah untuk menggambarkan suatu tahapan penyelesaian masalah secara sederhana, terurai, rapih dan jelas dengan menggunakan simbol-simbol yang standar. Tahap penyelesaian masalah yang disajikan harus jelas, sederhana, efektif dan tepat. Dalam penulisan ***flowchart*** dikenal dua model, yaitu ***Sistem Flowchart*** dan ***Program Flowchart***.

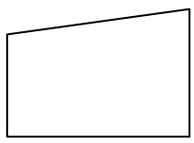
2.2.1. Sistem Flowchart

Sistem flowchart merupakan diagram alir yang menggambarkan suatu sistem peralatan komputer yang digunakan dalam proses pengolahan data serta hubungan antar peralatan tersebut. Sistem flowchart tidak digunakan untuk menggambarkan urutan langkah memecahkan masalah, tetapi hanya untuk menggambarkan prosedur dalam sistem yang dibentuk.

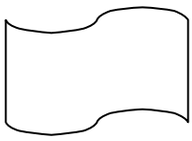
Simbol-simbol yang telah banyak digunakan pada penggambaran sistem flowchart adalah :



Pita Magnetik

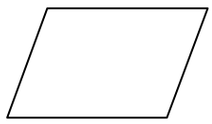


Kartu Plong / Keyboard

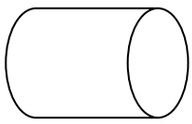


Punched Paper Tape

On Line Storage / VDU



Input / Output

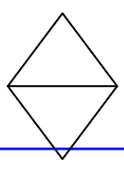


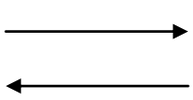
Magnetic Drum

Process

Magnetic Disk

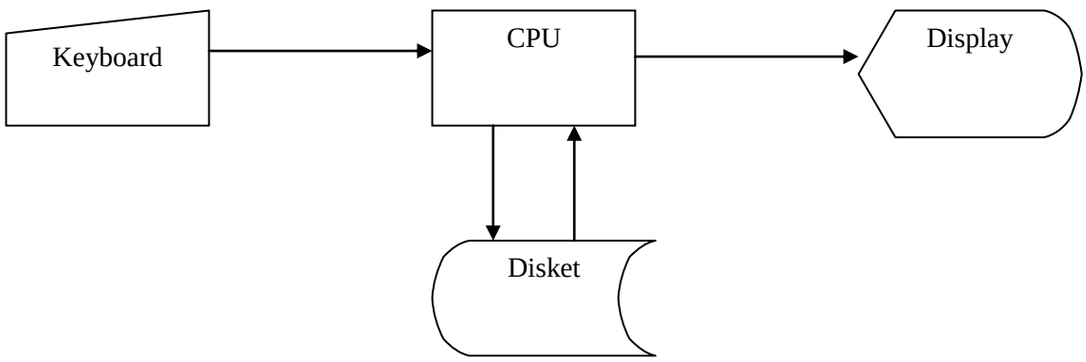
Off Line Storage





Proses Sortir Proses Merge Arus

Contoh penggunaan Sistem Flowchart :



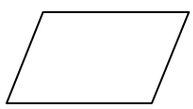
2.2.2. Program Flowchart

Program flowchart merupakan diagram alir program yang menggambarkan urutan logika dari suatu prosedur pemecahan masalah .

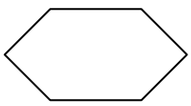
Dalam menggambarkan program flowchart telah tersedia simbol-simbol standar.

Baik program flowchart ataupun sistem flowchart dapat menambahkan simbol-simbol yang baru dengan syarat harus menambahkan simbol tersebut di dalam kamus simbol yang digunakan beserta keterangannya.

Simbol-simbol yang standar digunakan dalam menggambarkan program flowchart adalah :

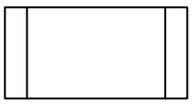


Input/ Output

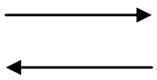


Inisialisasi Pemberian Nilai awal

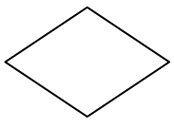
Proses



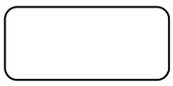
Keterangan



Arus



Pengujian Pilihan

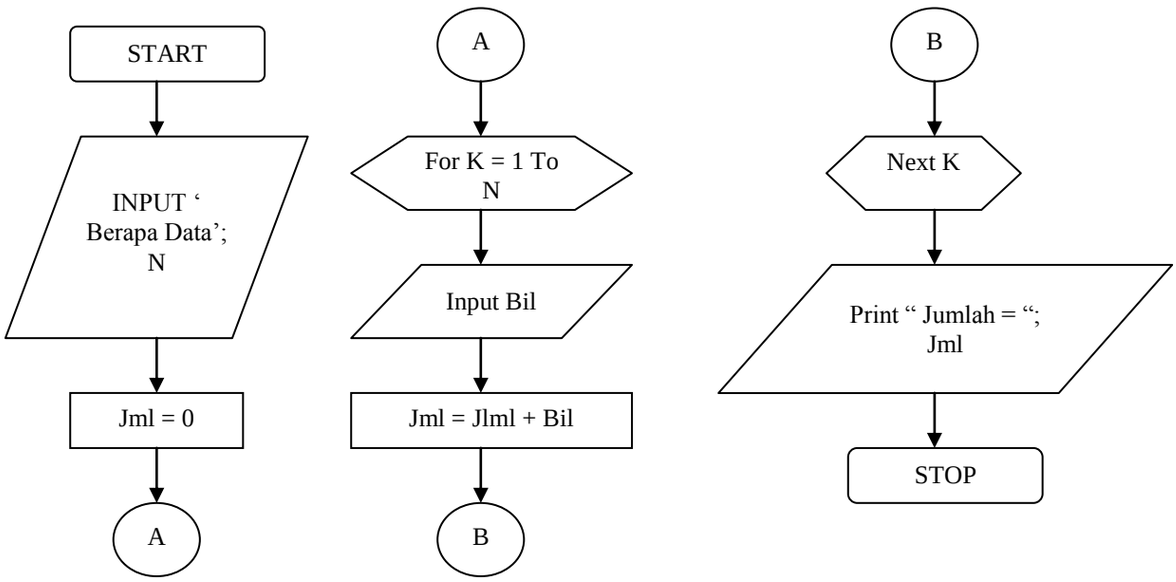


Awal / Akhir Program

Konektor dalam satu halaman

Konektor untuk halaman berbeda

Contoh penggunaan Program Flowchart :

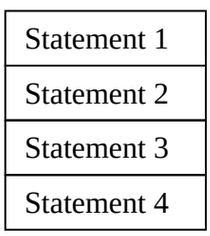


2.3. Nassi Schneiderman (Ns Diagram)

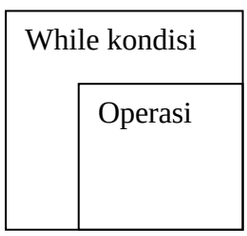
Selain dengan menggunakan flowchart, penggambaran algoritma dapat juga dilakukan dengan menggunakan flowchart Nassi Schneiderman. Pada dasarnya flowchart ini adalah flowchart yang terstruktur karena tanpa menggunakan tanda panah untuk proses pengulangannya.

Simbol-simbol yang digunakan dalam flowchart Nassi Scheneiderman adalah ;

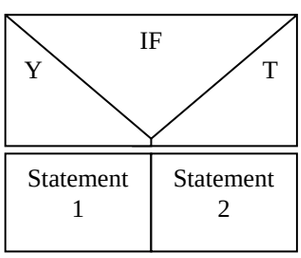
Sequence Control

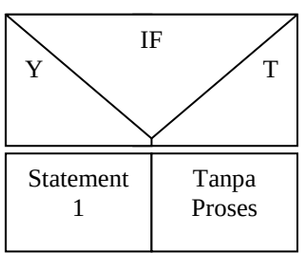


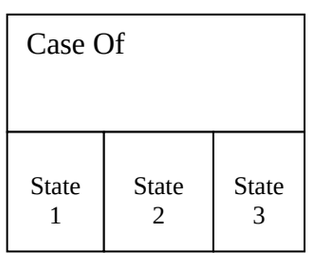
Looping Control



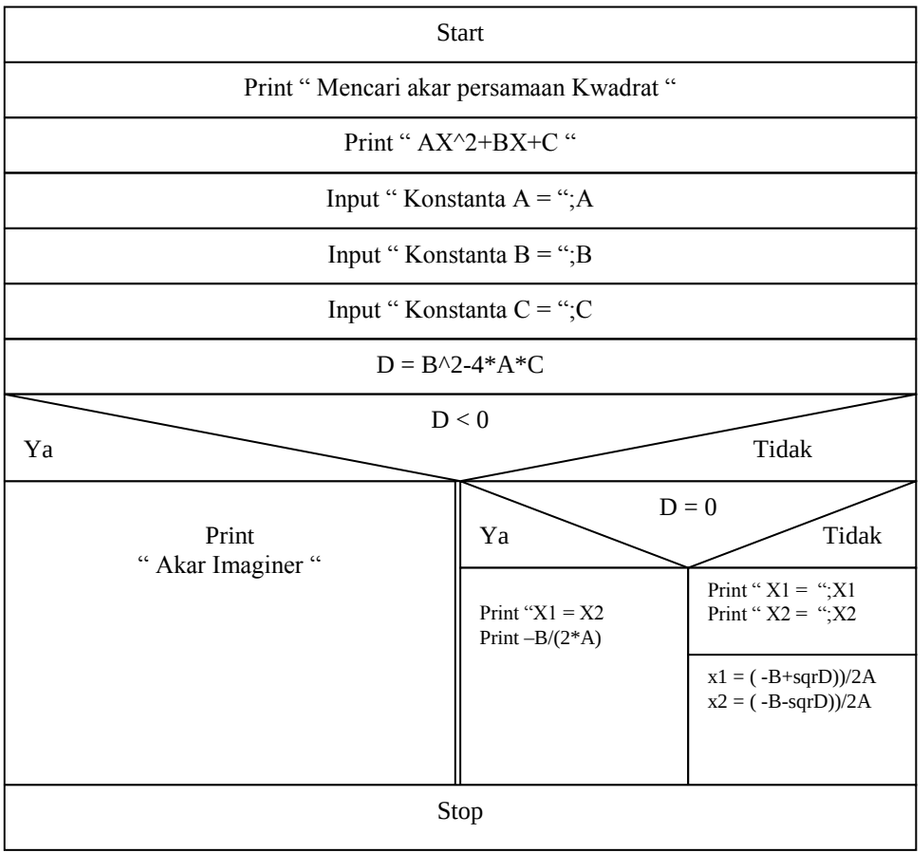
Selection Control







Contoh penggunaan flowchart Nassi Schneiderman dalam menyelesaikan persoalan akar persamaan kwadrat:



2.4. Pseudocode

Pseudocode adalah kode yang mirip dengan kode pemrogramman yang sebenarnya. Pseudocode berasal dari kata Pseudo yang mempunyai arti mirip atau imitasi atau menyerupai, dan code berarti program. Pseudocode ditulis berdasarkan pada bahasa pemrogramman BASIC, PASCAL, atau C sehingga lebih tepat digunakan untuk menggambarkan algoritma program yang akan dikomunikasikan kepada programmer yang menngunakan program tsb.

Dalam penulisan pseudocode dikenal struktur penulisan algoritma seperti sequence structure, selection structure dan looping structure.

Contoh algoritma untuk membuat Menu adalah seperti di bawah ini :

```
MENU PROGRAM
Membuat Arsip
Simpan Arsip
Hapus Arsip
Keluar dari Menu
Ketikan nomor Pilihan Menu : _
```

```
Algorithm Name : MENU
Kamus Lokal:
        Pil : integer [1..4] { nomor pilihan menu}
Algoritma
        Repeat
                Output (‘ MENU PROGRAM ’)
                Output (‘ 1. Membuat Arsip ’)
                Output (‘ 2. Simpan Arsip’)
                Output (‘ 3. Hapus Arsip’)
                Output (‘ 4. Keluar dari Menu’)
                Output (‘ Ketikan Nomor pilihan Menu :’)
                Input (pil)
                   Depend on pil
                       Pil=1 : output (‘ Anda Memilih menu Membuat Arsip ‘)
                       Pil=2 : output (‘ Anda Memilih menu Simpan Arsip ‘)
                       Pil=3 : output (‘ Anda Memilih menu Hapus Arsip ‘)
                       Pil=4 : output (‘ Anda Memilih menu Keluar dari menu‘)
                   Enddepend
        Until pil=4
```

Contoh :

Algoritma untuk menentukan bilangan tersebsar dari x, y, z.

```
IF x > y then
        IF x > z then
                Tulis x sebagai bilangan terbesar
        Else
                Tulis z sebagai bilangan terbesar
        Endif
Else
        IF y > z then
                Tulis y sebagai bilangan terbesar
        Else
                Tulis z sebagai bilangan terbesar
        Endif
Endif
```

Karena bahasan selanjutnya akan menggunakan model prosedural, maka bagian data dan bagian intruksi dipisahkan tempatnya. Pada dasarnya algoritma terbagi dalam 3 bagian yaitu :

Bagian Kepala (Head), **Bagian Deklarasi** dan **Bagian Deskripsi** algoritma. Setiap bagain disertai dengan komentar untuk memperjelas maksud teks yang dituliskan. Komentar adalah teks yang diapit oleh pasangantanda kurung kurawal { }, ( ).

2.4.1. Bagian Kepala Algoritma (Head)

Bagian kepala dalah bagian yang berisi nama algoritma dan penjelasan (spesifikasi) tentang algoritma tersebut.

Contoh :

Algoritma LUAS_LINGKARAN

(Menghitung luas lingkaran untuk ukuran jari-jari tertentu. Algoritma menerima masukan jari-jari, menghitung luasnya, lalu mencetak luas lingkaran ke piranti keluaran)

2.4.2. Bagian Deklarasi

Bagian ini mendeklarasikan semua nama yang digunakan dalam algoritma. Nama tersebut dapat merupakan nama Konstanta, nama variable, nama tipe, nama prosedur atau nama fungsi.

DEKLARASI

{ nama tetapan }

conts Npeg = 100 (jumlah pegawai)
conts phi = 3,14 (nilai phi)

{ nama variable}
c : character
q : bolean

function APAKAH_A(input c : cahar) → boolean
(menegmbalikan nilai True bila c adalah karakter ‘A’, atau false jika sebalikanya)

2.4.2. Bagian Dekripsi

Bagian ini adalah bagian yang berisi uraian langkah-langkah.penyelesaian masalah.

Contoh

DESKRIPSI

Algoritma MAKSIMUM
(Menentukan bilangan terbesar dari 3 buah bilangan bulat)

DEKLARASI
A, B, C : integer

DESKRIPSI
Read (A, B, C)
IF A > B **then**
    IF A > C **then**
        Write (‘Bilangan terbesar adalah ‘, A)
    Else
        Write (‘Bilangan terbesar adalah ‘, C)
    Endif
Else
    IF B > C **then**
        Write (‘Bilangan terbesar adalah ‘, B)
    Else
        Write (‘Bilangan terbesar adalah ‘, C)
    Endif
Endif

Contoh Algoritma lengkap :

Tugas :

1. Buatlah Algoritma dengan menggunakan Flowchart, Ns Diagram dan Pseudocode untuk menentukan banyaknya bilangan ganjil dan bilangan genap yang lebih kecil dari n (nilai yang diinputkan).

BAB III

ARRAY / LARIK

Tes Kemampuan Awal

1. Apa yang anda ketahui array satu dimensi dan array dua dimensi ?
2. Buatlah algoritma untuk mengisi dan menampilkan sebuah array serta menghitung jumlah dari seluruh isi elemen yang diinputkan.
3. Buatlah algoritma untuk mengisi dan menampilkan sebuah array serta menampilkan elemen yang ganjil saja atau yang genap saja beserta jumlahnya menurut pembagiannya.
4. Buatlah Algoritma untuk menghitung jumlah bilangan yang terletak pada garis diagonal sebuah matrik
5. Buatlah Algoritma untuk menjumlahkan dua buah matrik dengan menggunakan array dua dimensi !
6. Buatlah Algoritma untuk mengkailkan dua buah matrik dengan menggunakan array dua dimensi !

3.1. Array / Larik Satu Dimensi

3.1.1. Definisi Array / Larik Satu Dimensi

Array/Larik adalah struktur data yang mengacu pada sebuah atau sekumpulan elemen melalui indeks. Elemen Array/Larik dapat diakses langsung melalui indeksnya. Indek Array / Larik harus bertipe data yang menyatakan keterurutan, misalnya integer atau karakter.
Array/Larik yang bernama A dengan delapan buah elemen dapat dibayangkan secara logika sebagai kumpulan kotak yang berturutan

A

1	2	3	4	5	6	7	8

Tiap kotak pada Array/Larik tersebut diberi indeks integer 1,2,3,…,8. Tiap elemen Array/Larik ditulis dengan notasi :

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

Angka di dalam kurung siku menunjukan indeks dari Array/Larik.
Karena tiap elemen Array/Larik menyatakan tempat di memori, maka sebagaimana sifat memori adalah tempat penyimpan sementara, maka isi Array/Larik akan hilang jika komputer dimatikan.

4.1.2. Pendefinisian Array / Larik Satu Dimensi dalam inisialisasi

Array/Larik adalah struktur data yang statis, artinya jumlah elemen Array/Larik harus sudah diketahui sebelumnya. Jumlah elemen Array/Larik tidak dapat dirubah, ditambah atau dikurangi selama pelaksanaan program.
Mendefinisikan Array/Larik di dalam definisi variable berrati :
- Mendefinisikan banyaknya elemen Array/Larik
- Mendefinisikan tipe elemen Array/Larik

Mendefinisikan banyaknya elemen Array/Larik berarti memesa sejumlah tempat di meori, dan memori akan mengalokasikan tempat sebanyak elemen larik yang bersangkutan.
Contoh pendefinisian Array/Larik di dalam definisi variable :

```
Var
    mahasiswa    : array[1..100] of integer
```

Ini berarti bahwa nama variable array mahasiswa bertipe integer dan mempunyai elemen maksimum sebanyak 100 elemen.
Seperti halnya variable tunggal, tiap elemen Array/Larik dapat diisi melalui penugasan atau pembacaan dari input device.

3.1.3. Menginisialisasi Array / Larik Satu Dimensi

Menginisialisasi Array/Larik adalah memberi harga awal sebuah elemen Array/Larik. Inisialisasi kadang-kadang diperlukan, misalnya mengosongkan elemen Array/Larik sebelum dipakai untuk proses tertentu.

Mengosongkan Array/Larik bertipe numerik artinya mengisi elemen Array/Larik dengan nilai 0, jika bertipa karakter maka artinga mengisi elemen Array/Larik dengan spasi.

Contoh pemrosesan penginisialisasian elemen Array/Larik :

Penginisialisasian_elemen_ Array_Larik

Kamus Lokal :

A : array[1..100] of integer

I, N : integer

Algoritma :

Input (N)

For I = 1 to N do

A[I] ← 0

Endfor

3.1. 4. Pemrosesan Array / Larik Satu Dimensi

Elemen Array/Larik tersusun secara beruntun. Oleh karena itu, elemennya diproses secara beruntun melalui indeks yang terurut, asalkan indeks tersebut sudah terdefinisi. Pemrosesan dilakukan mulai pada elemen pertama Array/Larik secara berurutan sampai elemen terakhir.

Contoh pemrosesan pengisian elemen Array/Larik :

Pengisian_Umum_elemen_ Array_Larik

Kamus Lokal :

A : array[1..100] of integer

I, N : integer

Algoritma :

Input (N)

For I = 1 to N do

(A[I])= (A[I])+5

Endfor

Proses adalah aksi yang dilakukan terhadap elemen Array/Larik. Proses dapat berupa penugasan, pembacaan, penulisan atau manipulasi lainnya.

3.1.5. Input Elemen Array / Larik Satu Dimensi dari Input Device

Selain dengan penugasan, elemen Array/Larik dapat diisi nilainya dari input device dengan menggunakan perintah **INPUT**.

Contoh pengisian nilai Array/Larik dengan menggunakan perintah input.

Pengisian_elemen_ Array_Larik
Kamus Lokal :
    A : array[1..100] of integer
    I, N : integer
Algoritma :
    Input (N)
    For I = 1 to N do
        Input(A[I])
    Endfor

Seringkali tidak seluruh elemen Array/Larik yang didefinisikan digunakan dalam pemrosesan. Bila Array/Larik A didefinisikan 100 elemen, mungkin tidak seratus elemen yang dipakai. Banyaknya elemen Array/Larik yang dipakai disebut *jumlah elemen efektif*. Jumlah elemen efektif disimpan didalam variable tertentu, misalnya **`Nef`**. Kalau demikian, algoritma pengisian elemen Array/Larik diubah menjadi sebagai prosedur di bawah ini :

Procedure ISI_ARRAY(output A larik; Output `Nef` : integer)
Kamus Lokal :
    A : array[1..100] of integer
    I : integer
Algoritma :
    Input (Nef)
    For I = 1 to Nef do
        Input(A[i])
    Endfor

Algoritma Utama :

{**Mengisi_Array**}
ISI_ARRAY(A,Nef)

3.1.6. Output Elemen Array / Larik Satu Dimensi ke Output Device

Isi elemen array/larik dapat dicetak ke output device dengan perintah **Output**.

Contoh algoritma pencetakan :

Pencetakan_elemen_ Array_Larik

Kamus Lokal :

A : array[1..100] of integer

I, N : integer

Algoritma :

For I = 1 to N do

Output(A[i])

Endfor

3.1.7. Mencari Harga Tertentu di dalam Array / Larik Satu Dimensi

Pencarian beruntun adalah proses mengunjungi elemen array/larik dari elemen pertama, kemudian membandingkan dengan nilai yang dicari. Jika sama, maka pencarian sukses. Jika belum sama, maka akan melanjutkan pada elemen selanjutnya. Proses selanjutnya dilakukan sampai elemen yang dicari ditemukan atau sampai elemen array/larik habis.

Misalkan array A [1..8] telah berisi harga bilangan bulat :

A

1	2	3	4	5	6	7	8
56	45	12	34	11	24	98	11

Misalkan yang dicari adalah : X = 11

Pemeriksaan akan dilakukan terhadap 56, 45, 12, 34, 11.

X ditemukan pada elemen ke 5, jadi nilai X ← a [5].

Di dalam array A terdapat dua elemen 11, namun demikian yang diperhitungkan hanya elemen yang terdahulu ditemukan (dalam hal ini elemen yang ke 5).

Algoritma pencarian harga tertentu :

Procedure CARI_ARRAY(InputA : Array; Input Nef : integer; Input X : inetger; Output _idx : integer)

Kamus Lokal :

I : Integer

Algoritma :

I ← 1

While (I < Nef) and (A[I] ≠ X) do

I := I + 1

Endwhile (I := Nef) or (A[I] = X)

If A[I] = X then { X ditemukan }

Idx ← I { idx = I jika ditemukan}

Else

```
        Idx ← 9999 { idx = 9999 jika tidak ditemukan}
    Endif
```

3.1.8. Mencari Harga Maksimum di dalam Array / Larik Satu Dimensi

Proses pencariannya sama dengan proses pencarian umum, pada saat mengunjungi elemen yang pertama, elemen tersebut disimpan dalam nama variable tunggal. Kemudian variable tunggal tersebut dibandingkan dengan isi dari elemen array/larik selanjutnya. Apabila elemen array/larik yang dikunjungi lebih besar dari nilai variable tunggal, maka nilai elemen array/larik tersebut mengganti isi variable tunggal. Demikian untuk elemen selanjutnya. Jika elemen yang dikunjungi lebih kecil, maka akan dilakukan proses kunjungan ke elemen array/larik selanjutnya.

Bila elemen sudah terurut naik, maka pencarian sudah selesai pada elemen terakhir, sedang untuk elemen yang menurun, maka pencarian sudah selesai pada elemen yang pertama.

Bila elemen array tersusun acak, maka proses kunjungan akan dilakukan ke seluruh elemen.

```
Procedure MAKS_ARRAY ( input  A     : array;          input  Nef   : integer ;
                       Output maks  : integer ;  Output       Imaks : integer  )
Kamus Lokal :
    I     : Integer

Algoritma :
    Maks ← -9999
    For I ← 1 to Nef do
        If A[I] > maks then
            Maks  ← A [I]
            Imaks ← i
        Endif
    Endfor
```

Algoritma ini harga maksimum awalnya diinisialisasikan dengan nilai – 9999. Algoritma ini hanya akan benar apabila elemen array bernilai positif. Apabila bernilai negatif, hanya akan benar selama elemen array lebih besar dari –9999. Apabila ketentuan tersebut tidak dipenuhi, maka algoritma akan salah. Alternatif algoritma agar dapat digunakan dengan tanpa pembatasan adalah seperti di bawah ini :

```
Procedure MAKS_ARRAY ( input  A     : array;          input  Nef   : integer ;
                       Output maks  : integer ;  Output       Imaks : integer  )
```

Kamus Lokal :

I : Integer

Algoritma :

```
Maks ← A[I]
For I ← 2 to Nef do
    If A[I] > maks then
        Maks ← A [I]
        Imaks ← i
    Endif
Endfor
```

3.1.9. Mencari Harga Minimum di dalam Array / Larik Satu Dimensi

Proses pencariannya sama dengan proses pencarian umum, pada saat mengunjungi elemen yang pertama, elemen tersebut disimpan dalam nama variable tunggal. Kemudian variable tunggal tersebut dibandingkan dengan isi dari elemen array/larik selanjutnya. Apabila elemen array/larik yang dikunjungi lebih kecil dari nilai variable tunggal, maka nilai elemen array/larik tersebut mengganti isi variable tunggal. Demikian untuk elemen selanjutnya. Jika elemen yang dikunjungi lebih besar, maka akan dilakukan proses kunjungan ke elemen array/larik selanjutnya.

Bila elemen sudah terurut naik, maka pencarian sudah selesai pada elemen pertama, sedang untuk elemen yang menurun, maka pencarian sudah selesai pada elemen yang terakhir.

Bila elemen array tersusun acak, maka proses kunjungan akan dilakukan ke seluruh elemen.

Procedure MIN_ARRAY (input A : array; input Nef : integer ;
Output min : integer ; Output Imin : integer)

Kamus Lokal :

I : Integer

Algoritma :

```
Min ← 9999
For I ← 1 to Nef do
    If A[I] < maks then
        Min ← A [I]
        Imin ← i
    Endif
Endfor
```

Algoritma ini harga maksimum awalnya diinisialisasikan dengan nilai 9999. Algoritma ini hanya akan benar apabila elemen array bernilai negatif. Apabila bernilai positif, hanya akan

benar selama elemen array lebih kecildari 9999. Apabila ketentuan tersebut tidak dipenuhi, maka algoritma akan salah. Alternatif algoritma agar dapat digunakan dengan tanpa pembatasan adalah seperti di bawah ini :

```
Procedure MIN_ARRAY (   input  A      : array;     input  Nef    : integer ;
                        Output min    : integer ;  Output Imin   : integer  )
Kamus Lokal :
        I       : Integer
Algoritma :
        Min ← A[I]
        For I ← 2 to Nef do
                If A[I] < min then
                        Min  ← A [I]
                        Imin ← i
                Endif
        Endfor
```

3.2. Array / Larik Dua Dimensi

3.2.1. Definisi Array / Larik Dua Dimensi

Array/Larik Dua dimensi adalah struktur data yang sama dengan array satu dimensi, yaitu mengacu pada sebuah atau sekumpulan elemen melalui indeks. Elemen Array/Larik dapat diakses langsung melalui indeksnya. Indek Array / Larik harus bertipe data yang menyatakan keterurutan, misalnya integer atau karakter.

Array/Larik Dua dimensi yang bernama A dengan delapan buah elemen dapat dibayangkan secara logika sebagai kumpulan kotak yang berturutan

A

x,y	x,y+1	x,y+2	x,y+3	x,y+4	x,y+5
X+1,y	X+1,y+1	X+1,y+2	X+1,y+3	X+1,y+4	X+1,y+5
X+2,y	X+1,y+2	X+2,y+2	X+2,y+3	X+2,y+4	X+2,y+5
X+3,y	X+1,y+3	X+3,y+2	X+2,y+3	X+3,y+4	X+3,y+5
X+4,y	X+1,y+4	X+4,y+2	X+2,y+3	X+4,y+4	X+4,y+5
X+5,y	X+1,y+5	X+5,y+2	X+2,y+3	X+5,y+4	X+5,y+5

Tiap kotak pada Array/Larik dua dimensi tersebut diberi indeks integer [1…5,1…5] Tiap elemen Array/Larik dua dimensi ditulis dengan notasi :

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

Angka di dalam kurung siku menunjukan indeks dari Array/Larik dua dimensi.

3.2.2. Pendefinisian Array / Larik Dua Dimensi dalam inisialisasi

Array/Larik dua dimensi adalah struktur data yang statis, artinya jumlah elemen Array/Larik dua dimensi harus sudah diketahui sebelumnya. Jumlah elemen Array/Larik dua dimensi tidak dapat dirubah, ditambah atau dikurangi selama pelaksanaan program.
Mendefinisikan Array/Larik dua dimensi di dalam definisi variable berrati :
- Mendefinisikan banyaknya elemen Array/Larik dua dimensi
- Mendefinisikan tipe elemen Array/Larik dua dimensi

Mendefinisikan banyaknya elemen Array/Larik dua dimensi berarti memesan sejumlah tempat di meori, dan memori akan mengalokasikan tempat sebanyak elemen larik dua dimensi yang bersangkutan.
Contoh pendefinisian Array/Larik dua dimensi di dalam definisi variable :

```
Var
        mahasiswa      : array[1..100, 1..15] of integer
```

Ini berarti bahwa nama variable array mahasiswa bertipe integer dan mempunyai elemen maksimum sebanyak 100 elemen dan masing elemen tersebut mempunyai kapasitas maksimum data yang dpat disimpan sebanyak 5 buah.
Seperti halnya variable tunggal, tiap elemen Array/Larik dapat diisi melalui penugasan atau pembacaan dari input device.

3.2.3. Menginisialisasi Array / Larik Dua Dimensi

Menginisialisasi Array/Larik dua dimensi adalah memberi harga awal sebuah elemen Array/Larik dua dimensi. Inisialisasi kadang-kadang diperlukan, misalnya mengosongkan elemen Array/Larik dua dimensi sebelum dipakai untuk proses tertentu.
Mengosongkan Array/Larik dua dimensi bertipe numerik artinya mengisi elemen Array/Larik dua dimensi dengan nilai 0, jika bertipa karakter maka artinga mengisi elemen Array/Larik dua dimensi dengan spasi.

Contoh pemrosesan penginisialisasian elemen Array/Larik :

```
Penginisialisasian_elemen_ Array_Larik
Kamus Lokal   :
        A : array[1..100, 1..15] of integer
        I, J,X,Y : in      teger

Algoritma       :
        Input ( X )
```

```
Input ( Y )
For I = 1 to X do
        For J = 1 to Y do
                A[I,J] ← 0
        Endfor
Endfor
```

3.2.4. Pemrosesan Array / Larik Dua Dimensi

Elemen Array/Larik tersusun secara beruntun. Oleh karena itu, elemennya diproses secara beruntun melalui indeks yang terurut, asalkan indeks tersebut sudah terdefinisi. Pemrosesan dilakukan mulai pada elemen pertama Array/Larik secara berurutan sampai elemen terakhir.

Contoh pemrosesan pengisian elemen Array/Larik :

Pengisian_Umum_elemen_ Array_Larik

```
Kamus Lokal :
        A : array[1..100,1..5] of integer
        I, J,X,Y : integer
Algoritma :
        Input ( X )
        Input ( Y )
        For I = 1 to X do
                For J = 1 to Y do
                        A[I,J] ← A[I,J] + 5
                Endfor
        Endfor
```

Proses adalah aksi yang dilakukan terhadap elemen Array/Larik. Proses dapat berupa penugasan, pembacaan, penulisan atau manipulasi lainnya.

3.2.5. Input Elemen Array / Larik Dua Dimensi dari Input Device

Selain dengan penugasan, elemen Array/Larik dapat diisi nilainya dari input device dengan menggunakan perintah **INPUT**.

Contoh pengisian nilai Array/Larik dengan menggunakan perintah input.

Pengisian_elemen_ Array_Larik

```
Kamus Lokal  :
        A : array[1..100,1..5] of integer
        I, J,X,Y : integer
```

```
Algoritma       :
        Input ( X )
        Input ( Y )
        For I = 1 to X do
                For J = 1 to Y do
                        Input A[I,J]
                Endfor
        Endfor
```

Seringkali tidak seluruh elemen Array/Larik yang didefinisikan digunakan dalam pemrosesan. Bila Array/Larik A didefinisikan 100 elemen, mungkin tidak seratus elemen yang dipakai. Banyaknya elemen Array/Larik yang dipakai disebut *jumlah elemen efektif*. Jumlah elemen efektif disimpan didalam variable tertentu, misalnya **`Nef`**. Kalau demikian, algoritma pengisian elemen Array/Larik diubah menjadi sebagai prosedur di bawah ini :

Procedure ISI_ARRAY(output A larik; Output `Nef` : integer)

```
Kamus Lokal :
        A       : array[1..100,1..5] of integer
        I,J,X,Y : integer
Algoritma :
        Input ( X )
        Input ( Y )
        For I = 1 to X do
                For J = 1 to Y do
                        Input A[I,J]
                Endfor
        Endfor
```

Algoritma Utama :

{**Mengisi_Array**}

ISI_ARRAY(A,X,Y)

3.2.6. Output Elemen Array / Larik Dua Dimensi ke Output Device

Isi elemen array/larik dapat dicetak ke output device dengan perintah **Output**.

Contoh algoritma pencetakan :

Pencetakan_elemen_ Array_Larik

```
Kamus Lokal :
        A : array[1..100,1..5] of integer
        I, J,X,Y : integer
Algoritma :
```

```
Input ( X )
Input ( Y )
For I = 1 to X do
        For J = 1 to Y do
                Output A[I,J]
        Endfor
Endfor
```

Tugas :

1. Buatlah algoritma untuk menjumlahkan dua buah Matrik
2. Buatlah algoritma untuk mengalikan dua buah Matrik

BAB IV

FUNGSI DAN PROSEDUR

Tes Kemampuan Awal

1. Buatlah algoritma dengan program DELPHI untuk mencari banyaknya bilangan ganjil dan bilangan genap yang lebih kecil dari N (bilangan yang diinputkan) dengan menggunakan alat pseudocode serta menggunakan fungsi atau prosedur.
2. Buatlah algoritma dengan program DELPHI untuk mencari jumlah data sebanyak N (bilangan yang diinputkan) dengan nilai data yang bebas, dengan menggunakan alat pseudocode serta menggunakan fungsi atau prosedur.
3. Buatlah algoritma dengan program aplikasi DEKPHI untuk mencari banyaknya bilangan ganjil dan genap dari sejumlah data bilangan yang diinputkan dengan nilai data yang bebas, gunakan alat pseudocode serta fungsi atau prosedur.

4.1. Fungsi

Fungsi adalah modul program yang mengembalikan (return) sebuah nilai. Dalam matematika, anda sudah mengenal fungsi seperti ini :

f (x) = x^2 +3x – 5
g (t) = 2t – 3
h (x,y,z) = 2xy2 + 3yz + 10y

Pada contoh di atas, f,g dan h adalah fungsi, sedangkan x, t, x, y dan z adalah parameter fungsi yang bersangkutan. Dengan memberikan parameter, maka nilai fungsinya dapat dihitung.

4.1.1. Deklarasi Fungsi

Dalam algoritma yang aplikasi programnya menggunakan PASCAL, notasi untuk pendeklarasian fungsi adalah seperti di bawah ini :

```
Function name NAMAFUNGSI ( parameter masukan )
Kamus Lokal :
Algoritma :
        Awal
                :
        Akhir
        Return hasil
```

Parameter yang didefinisikan pada bagian judul fungsi dinamakan parameter **formal**. Parameter formal boleh tidak ada. Jika ada, parameter formal harus berupa nama variable. Semua nama yang didefinisikan dalam bagian variable, berlaku lokal (hanya di dalam daerah fungsi).

Contoh fungsi :
Untuk mendapatkan nilai F(x) = x^2 + 3x – 5, x $\in$ R

```
Function Mencari_F(Y:real ) → real
Kamus Lokal :
        x : real
Algoritma :
        Y ← x * x + 3 * x – 5
        Return y
```

4.1.2. Pemanggilan Fungsi

Fungsi dipanggil dengan menyebutkan nama beserta parameternya (jika ada).

NAMAFUNGSI (parameter)

Nama parameter yang disertakan pada waktu pemanggilan disebut parameter **aktual**. Parameter aktual dapat berupa konstanta, nama konstanta atau nama variable dengan syarat kesemuanya harus sudah didefinisikan terlebih dahulu tipe dan nilainya .

Hal-hal yang perlu diperhatikan dalam pemanggilan fungsi adalah :

1. Jumlah parameter aktual harus sama dengan jumlah parameter formal.
2. Tipe parameter aktual harus sama dengan tipe parameter formal.
3. Urutan parameter aktual harus sama dengan urutan parameter formal.
4. Nama-nama parameter aktual boleh tidak sama denganparameter formal.

Contoh pemanggilan fungsi F :

Output (Mencari_F(x))

4.2. Prosedur

Prosedur adalah modul program yang berisi rangkaian proses dan menghasilkan efek yang terdefinisi. Karena ada efek yang timbul inilah maka pada setiap prosedur harus didefinisikan keadaan awal (K.awal) sebelum rangkaian proses di dalam prosedur dilaksanakan dan keadaan akhir (K.akhir) setelah rangkaian proses dilaksanakan.

Seperti pada fungsi, prosedur diberi nama yang unik, disertai daftar parameter formal jika ada.

Parameter formal dalam prosedur dibedakan atas tiga jenis :

1. Parameter Input
2. Parameter Output
3. Parameter Input/Output

4.2.1. Deklarasi Prosedur

Notasi untuk pendeklarasian prosedur :

```
Procedure NAMAPROSEDUR
                    (Input/Output parameter formal )
Kamus Lokal   :
Algoritma     :
      Awal
            :
      Akhir
```

4.2.2. Pemanggilan Prosedur

Prosedur dipanggil dengan menyebutkan nama beserta parameter aktualnya nya (jika ada).

NAMAPROSEDUR(parameter aktual)

Pada waktu pemanggilan, terjadi relasi satu-satu antara parameter aktual dengan parameter formal. Harga atau nama parameter aktual “diterima” oleh paramater formal. Parameter berjenis input dapat berupa konstanta atau variable. (harganya sudah terdefinisi), sehingga pemanggilan dengan parameter berjenis input disebut juga pemanggilan dengan menggunakan nilai (call by value). Parameter berjenis output atau input/output harus beupa variable. Pada waktu pemanggilan yang dikirim ke parameter aktual bukan harga parameter aktual tetapi nama (acuan atau referensi) parameternya (termasuk harga yang dikandung), sehinga pemanggilan dengan parameter output atau input/output disebut juga pemanggilan dengan acuan (call by reference). Parameter berjenis input jika harganya dirubah dalam badan prosedur, harganya tidak berubah setelah pemanggilan, untuk parameter berjenis input/output harganya berubah setelah pemanggilan. Nama parameter aktual tidak boleh sama dengan parameter formal. Aturan yang berlaku di dalam fungsi berlaku juga dalam prosedur.

Contoh Prosedur :

```
Procedure HITUNG_RATA_RATA(input N : integer; output u : real )
{ K.awal : harga N terdefinisi K.akhir : u adalah rata-rata dari N buah bilangan }
Deklarasi:
        I,bil,toral         : integer
Deskripsi :
        Total ← 0           ( inisialisasi )
        I ← 1               ( inisialisasi pencacah )
        While I ≤ N do
                Input ( bil )
                Total ← total + bil
                i ← i + 1
        endwhile
        (K > N )
        u ← total / N
```

```
Algoritma RATA_RATA_BILANGAN_BULAT
( Program utama untuk menghitung rata-rata N buah bilangan bulat )

DEKLARASI
        N       : integer ( Banyaknya data bilangan bulat, N positif )
        U       : real ( nilai-rata-rata seluruh bilangan )

DESKRIPSI
Read N
Write (‘Menghitung rata-rata N buah bilangan bulat’)
HITUNG_RATA_RATA
Write (‘Nilai rata-rata = ’, u)
```

Pada prosedur dan program utama di atas, variable N dan u dideklarasikan di dalam bagian DEKLARASI program utama, karena itu N dan u bersifat global sehingga juga dikenal dan dapat digunakan dalam prosedur HITUNG_RATA_RATA. Sedangkan variable I, bil dan

total dideklarasikan di dalam bagian DEKLARASI prosedur HITUNG_RATA_RATA., karena itu I, bil dan total bersifat lokal dan hanya dikenal dalam prosedur HITUNG_RATA_RATA.

4.3. Nama Global dan Nama Lokal

Nama-nama (kostanta, variable, tipe, dll) yang dideklarasikan dalam bagian DEKLARASI sebuah prosedur hanya dikenal dalam badan prosedur yang bersangkutan. Hal ini biasanya dikatakan sebagai **variable lokal**. Untuk nama-nama yang dideklarasikan pada bagian Deklarasi program utama dapat dikenal di seluruh bagian program. Biasanya dinamakan sebagai **variable global**.

4.4. Menggunakan Fungsi atau Prosedur

Tidak ada aturan baku yang menyatakan apakah sebuah modul direalisasikan sebagai fungsi atau prosedur. Suatu fungsi dapat ditulis sebagai prosedur, namun hanya prosedur yang menghasilkan satu keluaran saja yang dapat dijadikan fungsi. Bila fungsi dinyatakan sebagai prosedur, maka perubahan yang dilakkan adalah pada tipe hasil. Hasil yang dikembalikan oleh fungsi dinyatakan sebagai parameter formal berjenis output.

BAB V

SORTING

Tes Kemampuan Awal

1. Sebutkan Jenis Sorting yang anda ketahui, terangkan bagaimana proses pengurutannya dengan memberikan sebuah contoh !
2. Nilai apa yang dicari untuk mendapatkan sebuah metode pengurutan yang paling baik ?

5.1. Sorting (Pengurutan)

Sorting (pengurutan) secara umum diartikan sebagai proses penyusunan kembali sekumpulan objek ke dalam suatu urutan / sekuaens tertentu berdasarkan kriteria pengurutan tertentu. Tujuan sorting adalah untuk memudahkan silakukannya proses *search* (pencarian) terhadap objek yang terdapat dalam kumpulan yang telah terurut tersebut.

Sebagai contoh, perhatikan proses mencari nomor telepon dalam sebuah buku telepon. Proses *search* ini menjadi sangat mudah karena nama- nama dalam buku telepon sudah terurut.

Jika kita memiliki ietem-item data : a_1,a_2 , a_3, … ,a_n, maka proses sorting mengubah urutan / mempertukarkan posisi item/item tersebut sehingga diperoleh a_{k1},a_{k2} , a_{k3}, … ,a_{kn}, sedemikian rupa sehingga, jika terdapat sebuah *ordering function* f, maka f($a_{k1)}$ < f(a_{k2} < f(a_{k3}) , … f(a_{kn}). Umumnya, fungsi ini berbentuk rumus komputasi, namun disimpan dalam bentuk komponen / field dari masing-masing item. Nlainya disebut key dari item tersebut.

Sebuah file berukuran n ialah kumpulan n buah itemn r(1), r(2), r(3), … , r(n). Setiap item file disebut record. Pada setiap record r (i) terdapat sebuah key k (i). Key ini biasanya adalah sebuah subfile dari keseluruhan record, baik dengan urutan menaik (ascending) maupun menurun (descending). Sebuah metode sort dikatakan stabil (stable) apabila untuk semua record i dan j dimana k(i) = k (j), jika r (i) mendahului r (j) pada file awal, maka r (i) juga mendahului r (j) pada file yang terurut. Dengan kata lain, urutan relatif dari item-item yang nilai key-nya sama tidak akan berubah di akhir proses pengurutan.

Proses sorting dapat dilakukan langsung terhadap record-record yang ada atau terhadap sebuah tabel bantu yang berisi pointer-pointer (disebut *sorting by address*). Sebagai contoh, pada gambar dibawah ditunjukan ditunjukan sebuah file dengan 5 record. Jika file tersebut diurutkan secara menaik (*increasing order*) terhadap key *numeric*-nya, maka hasilnya terlihat pada tabel sebelahnya. Dalam kasus ini record-recordnya sendiri telah diurutkan secara aktual.

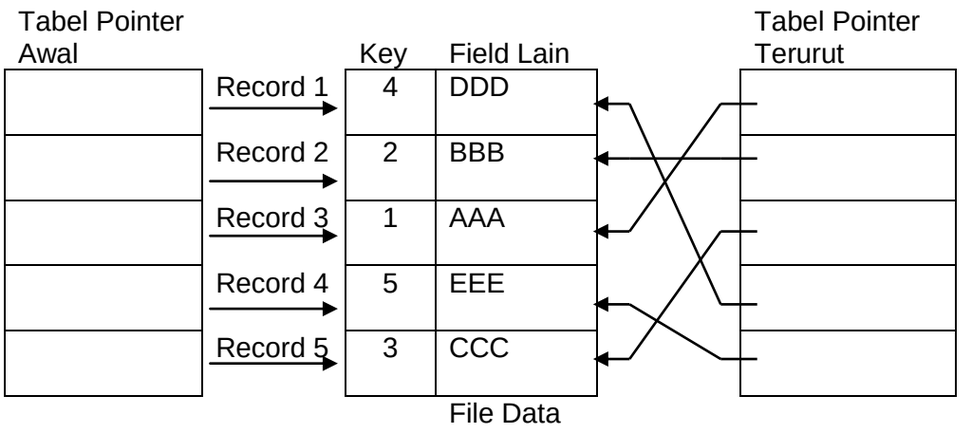
	Key	Field lain
Record 1	4	DDD
Record 2	2	BBB
Record 3	1	AAA
Record 4	5	CCC
Record 5	3	EEE

File

Key	Field lain
1	AAA
2	BBB
3	CCC
4	DDD
5	EEE

File

Namun apabila jumlah data / field yang tersimpan dalam masing-masing record pada *file* tersebut cukup besar, maka *overhead* waktu yang dibutuhkan untuk memindahkan data secra aktual akan sangat besar. Hal ini tidak dikehendaki dalam proses *sorting*. Dalam kasus ini, sebuah tabel pointer bantu dapat digunakan agar pada proses sorting yang dipindahkan adalah penunjukan pointer, bukan data aktual. Hal ini ditunjukan pada gambar di bawah ini.



Tabel di bagian tengah adalah file data dan tabel disisi kiri adalah tabel pointer awal. Entry pada posisi j dalam tabel pointer menunjuk ke record j. Pada proses sorting, entri-entri pada tabel pointer ini disesuaikan dengan penunjukannya, dan hasilnya ditunjukan pada tabel pointer di sisi kanan. Pointer pertama akhirnya menunjuk ke record ketiga dalam file, dan seterusnya. Perhatikan bahwa ada record yang dipindahkan.

Terdapat banyak metode yang digunakan untuk melakukan pengurutan data. Programmer harus memahami beberapa faktor penentu (konsiderasi) efisiensi yang saling berhubungan agar dapat memilih secara tepat metode sorting yang mana yang paling tepat diterapkan pada suatu masalah tertentu. Beberapa faktor penting yang menentukan efisiensi sebuah metode sorting adalah :

- Waktu yang dibutuhkan oleh programmer untuk mengkodekannya (membuat program sorting),
- Waktu komputasi yang dibutuhkan untuk mengeksekusi program sorting tersebut,
- Jumlah memori / storage yang dibutuhkan untuk menyimpan / mengeksekusi program sorting tersebut.

Requirement yang dituntut dari sebuah metode internal sorting adalah penggunaan kapasitas memori yang tersedia secara ekonomis. Dengan kata lain, pertukaran item-item data untuk memperoleh urutan yang benar harus dilakukan secara *in situ*. Metode yang memindahkan item data dari sebuah array A ke array B jelas tidak dikehendaki.

Ukuran yang baik untuk menilai efisiensi sebuah metode sorting adalah dengan menghitung jumlah perbandingan (*comparison*) key / data dan jumlah peretukaran (*interchange*) key / data yang dilakukan selama proses sorting. Kedua ukuran ini merupakan fungsi dari jumlah item data n yang akan diurutkan.

Metode sorting yang baik membutuhkan jumlah perbandingan dalam orde (n log n). Metode sorting sederhana (**straight method**) membutuhkan jumlah perbandingan dengan orde n^2. Namun ada tiga alasan mengapa **straight method** perlu ditinjau sebelum kita beranjak kepada metode-metode yang lebih cepat, yaitu :

- **straight method** merupakan cara tepat untuk menjelaskan karakteristik dari prinsip-prinsip utama dalam sorting,

- program untuk **straight method** dapat dimengerti dengan mudah dan tidak panjang sehingga kebutuhan *starage*-nya kecil
- walapun metode yang cepat membutuhkan jumlah operasi yang lebih sedikit, rincian operasi-operasi ini biasanya rumit. Jadi **straight method** lebih cepat bila diterapkan untuk jumlah data n yang kecil, namun tidak dianjurkan untuk jumlah data n yang besar.

4.2. NOTASI O BESAR (BIG O NOTATION)

Perhatikan tabel di bawah ini, Fungsi 0,01 n^2 + 10 n dikatakan berada dalam orde fungsi n^2 karena seiring dengan semakin besarnya nilai n, maka nilai fungsi tersebut akan semakin mendekati nilai n^2.

n	A = 0,01 n^2	b = 10 n	a + b	(a + b) / n^2
10	1	100	101	1,01
50	25	500	525	0,21
100	100	1.000	1.100	0,11
500	2.500	5.000	7.500	0,03
1.000	10.000	10.000	20.000	0,02
5.000	250.000	50.000	300.000	0,01
10.000	1.000.000	100.000	1.100.000	0,01
50.000	25.000.000	500.000	25.500.000	0,01
100.000	100.000.000	1.000.000	101.000.000	0,01
500.000	2.500.000.000	5.000.000	2.505.000.000	0,01

Duketahui dua buah fungsi f(n) dan g(n). fungsi f(n) dukatakan berada dalam orde g(n) atau f(n) adalah O(g(n)) jika tedapat bilangan bulat posistif a dan b sedemikian rupa sehingga f(n) $\leq$ a x g(n) untuk semua n $\geq$ b.

Sebagai contoh, jika f(n) = n^2 + 100 n dan g(n) = n^2, maka f(n) adalah O(g(n)) karena n^2 + 100 n $\leq$ $2n^2$ untuk semua n $\geq$ 100. Dalam kasus ini, a = 2 dan b = 100. Fungsi f(n) di atas adalah juga O(n^3) karena n^2 + 100 n $2n^3$ untuk semua n $\geq$ 8.

Jika diketahui sebuah fungsi f(n), maka terdapat banyak fungsi g(n) dimana fungsi f(n) adalah O(g(n)), dan untuk setiap fungsi g(n) tersebut terdapat banyak nilai a dan b yang memenuhi kriteria di atas.

Jika f(n) adalah O(g(n)) dan g(n) adalah O(h(n)), maka f(n) adalah O(h(n)). Sebagai contoh : n^2 + 100 n adalah O(n^2) dan n^2 adalah O(n^3) (untuk nilai a=b=1), maka n^2 + 100 n adalah O(n^3). Hal ini disebut sifat transitif.

Jika f(n) adalah sebuah fungsi konstanta, yaitu f(n) = c untuk semua nilai n, maka f(n) adalah O(1), karena dengan menggeser a = c dan b = 1, diperoleh bahwa c $\leq$ c x 1 untuk semua n $\geq$ 1.

Jika f(n) = c x g(n), dimana c adalah sebuah konstanta, maka f(n) adalah O(g(n)).

Berikut ini adalah beberapa fakta yang akan membentuk hirarki orde dari fungsi :

- c adalah O(1) untk sembarang konstanta c,
- c x $\log_k$ n adalah O(log n) untuk sembarang konstanta c dan k,

- c x n^k adalah O(n^k) untuk sembarang konstanta c dan k,
- c x n x $\log_k$ n adalah O(n log n) untuk sembarang konstanta c dan k,
- c x n^j x $\log_k$ n adalah O(n^k log n) untuk sembarang konstanta c, j dan k,
- c x n^j x ($\log_k$ n)^1 adalah O(n^k (log n)^1) untuk sembarang konstanta c, j, k dan l,
- c x n^k adalah O(d^n), namun d^n tidak O(n^k) untuk sembarang konstanta c dan k, dan d > 1.

Hirarki fungsi yang terbentuk dari fakta-fakta di atas adalah sebagai berikut, (tiap fungsi memiliki orde lebih rendah dari fungsi berikutnya) :

C, log n, (log n)^k, n, n(log n)^k, n^k, n^k (log n)^k, n^{k+1}, d^k.

Fungsi-fungsi yang ber-orde O(n^k) untuk setiap nilai k dikatakan berada dalam *polynomial order*, sedangkan fungsi-fungsi yang ber-orde O(d^n) untuk setiap d >1 tetapi tidak ber-orde O(n^k) untuk suatu nilai k dikatakan berada dalam *exponential order*.

Perbedaan antara fungsi polynomial-order dengan fungsi exponetial-order sangat penting. Sebuah fungsi exponential order yang sederhana sekalipun, seperti 2^n, akan bertambah jauh lebih cepat nilainya dibandingkan dengan polynomial-order, seperti n^k, terlepas dari berapapun nilai k.

Dengan menggunakan konsep orde di atas, berbagai metode sorting dapat dibandingkan dan mengklasifikasikannya dalam golongan “baik” atau “buruk”. Namun jika mengehendaki metode sorting yang “optimal”, yaitu yang berorde O(n) terlepas dari urutan data yang diinput, tidak akan ada metode yang dapat memenuhi hal tersebut.

Sebagian besar metode sorting yang akan ditinjau berikut ini mempunyai *time requirement* antara O(n log n) hingga O(n^2). Untuk ordeO(n log n), jika ururtan file bertambah 100 kali, maka *sorting time* akan berkurang dari 200 kali (dasar logaritnya adalah 10). Untuk orde O(n^2), jika ukuran file bertambah 100 kali, maka *sorting time* akan bertambah 10.000 kali.

4.3 INTERNAL SORT *BY EXCHANGE*

4.3.1. METODE BUBBLE SORT

Metoide Bubble Sort termasuk pada golonga Exchange Sort, dimana pertukaran tempat anatar dua item data menjadi karakteristik utama dari proses pengurutan. Metode bubble sort didasarkan pada prinsip perbandingan (*comparison*) dan pertukaran (*exchange*) pasanga item data yang berdekatan (*adjacent*) hingga semua data item terurut.

Keunggulan metoda Bubble sort adalah kemudahannya dipahami dan diprogram. Namun dari semua metode sort, metode ini termasuk pada metode yang paling tidak efisien.

Misalkan X adalah sebuah array bertipe integer berisi n buah data yang akan diurutkan guna mencapai kondisi :

X[i] ≤ X[j] untuk 1 ≤ j ≤ n

Metode Bubble sort menelusuri isi file secara sekuensial beberapa kali. Pada tiap langkah penelusuran (pass / iterasi), dilakukan pembandingan antara tiap item data dengan item

data berikutnya (successor-nya), atau X[1] dengan X[i+1], dan pertukaran posisi elemen data apabila urutannya tidak tepat.

Sebagi contoh, perhatikan nilai-nilai data berikut ini :

25 57 48 37 12 92 86 33

Diasumsikan bahwa hendaknya melakukan pengurutan secara menaik (*ascending*). Pada iterasi pertama, dilakukan perbandingan sebagai berikut :

X[1] dengan X[2] (25 dengan 57) → tidak terjadi pertukaran
X[2] dengan X[3] (57 dengan 48) → terjadi pertukaran
X[3] dengan X[4] (57 dengan 37) → terjadi pertukaran
X[4] dengan X[5] (57 dengan 12) → terjadi pertukaran
X[5] dengan X[6] (57 dengan 92) → tidak terjadi pertukaran
X[6] dengan X[7] (92 dengan 86) → terjadi pertukaran
X[7] dengan X[8] (92 dengan 33) → terjadi pertukaran

Jadi, setelah iterasi pertama, urutan nilai data adalah sebagai berikut :

25 48 37 12 57 86 33 **92**

Perhatikan setelah iterasi pertama, nilai data terbesar, yaitu 92, terletak pada posisinya yang seharusnya dalam array (posisi terakhir). Secara umum, data X[n-i+1] akan terletak pada posisi seharusnya setelah iterasi yang ke i.

Metode ini disebut "*bubble sort"* karena setiap nilai data secara perlahan-lahan "*bubbles*" (bergeser naik ke sisi kanan) ke posisinya yang tepat. Nilai data yang besar akan bergeser "lebih cepat". Setelah iterasi ke 2, urutan nilai data adalah ;

25 37 12 48 57 33 **86** **92**

Perhatikan bahwa 86 sebagai nilai data terbesar kedua sudah terletak pada posisi yang tepat.

Karena tiap iterasi menempatkan sebuah nilai data ke posisi yang seharusnya, maka sebuah file yang berisi n buah data membutuhkan tidak lebih dari n-1 iterasi.

Iterasi lengkapnya adalah sebagai berikut:

Awal	25	57	48	37	12	92	86	33
Iterasi 1	25	48	37	12	57	86	33	**92**
Iterasi 2	25	37	12	48	57	33	**86**	**92**
Iterasi 3	25	12	37	48	33	**57**	**86**	**92**
Iterasi 4	12	25	37	33	**48**	**57**	**86**	**92**
Iterasi 5	12	25	33	**37**	**48**	**57**	**86**	**92**
Iterasi 6	12	25	**33**	**37**	**48**	**57**	**86**	**92**
Iterasi 7	**12**	**25**	**33**	**37**	**48**	**57**	**86**	**92**

Sebelum menulis programnya, ada beberapa hal yang harus diperhatikan.

Pertama, karena semua nilai data pada posisi lebih besar dari atau sama dengan n-i+1 telah menempati posisi seharusnya setelah iterasi I, maka nilai-nilai data tersebut tidak perlu diikutsertakan lagi dalam proses perbandingan pada iterasi-iterasi selanjutnya. Jadi pada

ietrasi pertama dilakukan n-1 perbandingan, pada iterasi ke 2 dilakukan n-2 perbandingan, dan pada interaksi ke-(n-1) hanya dilakukan perbandingan (yaitu antara X[1] dan X[2].

Kedua. Diatas telah disebutkan bahwa dibutuhkan maksimal n-1 interasi untuk mengurutkan array berukuran n data. Namun, pada contoh di atas, array telah terurut membesar setelah interasi ke-5, sehingga interasi terakhir sebenarnya tidak perlu dilakukan (iterasi ke-6 dan ke-7). Untuk menghilangkan interasi seperti ini, kita harus dapat mendeteksi apakah sebuah array telah terurut atau belum.

Apabila array telah terurut, maka tidak terjadi pertukaran data apapun pada interasi tersebut. Jika kondisi ini dicapai, tidak perlu dilakukan interasi berikutnya. Dengan cara seperti ini, jika file dapat diurutkan dengan interasi kurang dari n-1, maka pada interasi terakhir tidak dilakukan pertukaran data. Pada contoh di atas, proses dilakukan hingga interasi ke-5 masih terjadi pertukaran data.

Kode program untuk metode bubble sort dalam bahasa pascal diberikan di bawah ini.

```
Procedure Bubble Sort (var X : ArrayType : N : integer);
Var  pass, j, temp, : integer;
        Swap : boolean;
Begin
        Swap  : = true;
        Pass  : =  1;
        While (pass <= N-1) and (swap) do
                Bgin
                        Swap : + false;
                        For j : = 1 to N-pass do
                                If (X[j] > X[j+1]) then
                                Begin
                                        {lakukan swapping/pertukaran elemen]
                                        swap : = true;
                                        temp : = X[j];
                                        X[j]  : = X[j+1];
                                        X[j+1: = temp;
                                End;
                        Pass : = pass + 1;
                End;
End;
```

Berikut ini diberikan contoh kedua. Pada contoh ini, pergeseran terjadi kesisi kiri. Semakin kecil nilai data, “semakin cepat” pergeserannya. Pada interasi pertama, nilai data terkecil menempati posisi pertama dalam array; pada interasi kedua, nilai data terkecil kedua menempati posisi kedua dalam array; dan seterusnya.

Pada contoh ini, proses pengurutan dilakukan hingga interasi ke-5, karena tidak terjadi lagi pertukaran data pada interasi ini. Interasi lengkapnya adalah.:

Awal	44	55	12	42	94	18	6	67
Interasi 1	6	44	55	12	42	94	18	67
Interasi 2	6	12	44	55	18	42	94	67
Interasi 3	6	12	18	44	55	42	67	94
Interasi 4	6	12	18	42	44	55	67	94
Interasi 5	6	12	18	42	44	55	67	94

Jumlah perbandingan dilakukan pada interasi-i adalah n-i, jadi, jika dilakukan k kali interasi, jumlah total perbandingan yang dilakukan adalah (n-1) + (n-2) +...+ (n-k) = $(2kn-k^2-k)/2$. Rata-rata jumlah iterasi (k) adalah $O(n^2)$.

Jumlah pertukaran yang dilakukan tergantung pada awal data dalam array/file. Namun, jumlah pertukaran ini tidak mungkin lebih besar dari jumlah perbandingan. Jumlah pertukaran diperkirakan menghabiskan sebagian besar waktu eksekusi algoritma, bukan jumlah perbandingan. Dalam kasus terburuk, jumlah perbandingan dalam bubble sort adalah $(n^2-n)/2$ dan jumlah pertukaran adalah $(n^2-n)/2$. Jadi, baik jumlah perbandingan maupun jumlah pertukaran berorde $O(n^2)$.

Kelebihan bubble sort satu-satunya hanyalah kebutuhan akan memori tambahan kecil, yaitu hanya satu lokasi memori untuk menyimpan nilai sementara (template) dalam proses pertukaran data, dan ordenya menjadi O(n) dalam kasus data masukan sudah /hampir terurut.

Perbaikan terhadap metode bubble sort dilakukan dengan algoritma shaker sort. Penjelasan mengenai algoritma ini beserta contohnya dapat dilihat dalam [WIR76].

4.3.2. METODE QUICK SORT (*PARTITION EXCHANGE SORT*)

Metode quick sort, yang dibuat oleh C.A.R Hoare, didasarkan pada fakta bahwa pertukaran (exchange) mungkin dapat dilakukan antara 2 item data yang posisinya berjauhan agar proses pengurutan menjadi lebih efektif. Jadi, metode ini tergolong dalam kelompok exchange sort.

Diasumsikan X adalah sebuah array berisi N buah data yang akan diurutkan. Kita memilih sebuah elemen A dari posisi tertentu dalam array (misalnya A adalah elemen pertama; A = X[1]). Nilai A disebut elemen kunci atau pivot atau comparand. Semua elemen data dalam array X dibagi menjadi dua sedemikian rupa sehingga A diletakan pada posisi j dan kondisi-kondisi berikut dipenuhi :

- Setiap elemen data pada posisi 1 s.d j -1 nilainya lebih kecil dari atau sama dengan nilai pivot A.
- Setiap elemen data pada posisi j+1 s.d n nilainya lebih besar dari atau sama dengan nilai pivot A.

Perhatikan bahwa setiap kedua kondisi tersebut terpenuhi untuk nilai A dan j tertentu, maka A akan teteap terletak pada posisi j setelai array selesai diurutkan. Jika proses ini diulang untuk subarray X[1] – X [j-1], untuk subarray X[j+1] – X[N], dan seterusnya untuk subarray-subarray yang terbentuk dalam interasi-interasi berikutnya, maka hasil akhirnya adalah sebuah file yang terurut.

Berikut ini diberikan sebuah contoh. Urutan data awal dalam array adalah sbb.:

25 57 48 37 12 92 86 33

Nilai data pertama, yaitu 25, diletakan pada posisi yang seharusnya hasilnya adalah;

12 25 57 48 37 92 86 33

Pada titik ini, 25 terletak pada posisi yang tepat, yaitu X[2]. Setiap data yang letaknya sebelum posisi ke-2, yaitu 12, nilainya lebih kecil dari atau sama dengan 25, dan setiap data yang letaknya setelah posisi ke-2, yaitu 57, 48, 37, 92, 86, dan 33, nilainya lebih besar dari atau sama dengan 25.

Karena 25 telah terletak pada posisi yang benar, maka masalah awal telah dibagi (dekomposisi) menjadi masalah pengurutan dua buah subarray, yaitu :

[12] dan [57 48 37 92 86 33]

Karena sub array pertama berisi 1 nilai data, maka sub array tersebut telah terurut dan tidak perlu diproses. Untuk mengurutkan sebarray kedua, proses di atas diulangi dan subarray ini kemudian dibagi lagi menjadi dua subarray baru 57 diambil sebagai nilai pivot. Hasilnya adalah :

[48 37 33] 57 [92 86]

Hasil tiap tahap pengurutan selengkapnya adalah sbb .:

25	57	48	37	12	92	86	33
[12]	25	**[57**	**48**	**37**	**92**	**86**	**33]**
12	25	**[57**	**48**	**37**	**92**	**86**	**33]**
12	25	**[48**	**37**	**33]**	57	**[92**	**86]**
12	25	**[37**	**33]**	48	57	**[92**	**86]**
12	25	**[33]**	37	48	57	**[92**	**86]**
12	25	33	37	48	57	**[86]**	92
12	25	33	37	48	57	86	92

Algoritma untuk metode quick sort dapat dituliskan dengan mudah dalam bentuk prosedur rekursif. Dibawah ini diberikan algoritma untuk prosedur Quick(LB, UB) yang mengurutkan semua item data dalam array X diantara posisi X[LB] dan X[UB], dimana LB adalah batas bawah (lower bound) dan UB adalah batas atas (upper bound).

```
If (LB < UB) then
{jika LB ≥ UB, maka array telah terurut}
begin
        {pilih item-2 dari subarray sedemikian rupa sehingga salah satu item (misal X([LB])
        Berada pada posisi  X [j]
        (j merupakan parameter output) dan berlaku :
        1.   x [i] ≤ x [j] untuk LB ≤ i < j
        2.   x [i] ≥ x [j] untuk j < i  ≤ UB
        x [j] telah berada pada posisi yang seharusnya .}
        partition (LB, UB, j);
        {sort subarray antara x[LB dan x[j-1]}
        Quick (LB, j-1);
        {sort subarray antara x[j+1] dan x[UB]}
        Quick (j+1, UB);
End;
```

Algoritma rekursif untuk quick sort

Salah satu cara untuk melakukan pemilihan array (dalm prosedur partition) dengan efisien adalah sebagai berikut. Asumsikan A = X[LB] adalah item data yang akan dicari posisi sebenarnya. Dua buah pointer, yaitu up dan down, diinisialisasikan dengan batas bawah

dan batas kiri dari subarray. Pada setiap titik selama eksekusi, tiap item data pada posisi di atas / di kanan up nilainya lebih besar dari atau sama dengan nilai A, dan tiap item pada posisi di bawah /di kiri down nilainya lebih kecil dari atau sama dengan nilai A. Kedua pointer up dan down ini digeser mendekati satu sama lain dengan cara berikut :

- Langkah 1.
  Naikkan (increment) nilai pointer down satu posisi secara berulang-ulang hingga X[down] > A.
- Langkah 2.
  Turunkan (decrement) nilai pointer up satu posisi secara berulang-ulang hingga X[up] ≤ A.
- Langkah 3.
  Jika (up > down), pertukarkan posisi X[down] dengan X[up].

Proses diatas diulangi hingga kondisi pada langkah 3 tidak dipenuhi (up ≤ down). Pada titik ini X[up} dipertukarkan dengan X[LB] (yaitu pivot A), dan nilai j diset menjadi up. Berikut ini akan diberikan gambaran mengenai pross tersebut, dimana ditunjukan posisi dari up dan down.

Down →							up
25	57	48	37	12	92	86	**33**
	down						up
25	**57**	48	37	12	92	86	**33**
	down						← up
25	**57**	48	37	12	92	86	**33**
	down					← up	
25	**57**	48	37	12	92	**86**	33
	down				← up		
25	**57**	48	37	12	**92**	86	33
	down			up			
25	**57**	48	37	**12**	92	86	33
	down			up			
25	**12**	48	37	**57**	92	86	33
	down →			up			
25	**12**	48	37	**57**	92	86	33
		down		up			
25	12	**48**	37	**57**	92	86	33
		down		← up			
25	12	**48**	37	**57**	92	86	33
		down ← up					
25	12	**48**	37	57	92	86	33
	←	up,down					
25	12	**48**	37	57	92	96	33
	up	down					
25	**12**	**48**	37	57	92	86	33
	up	down					
12	**25**	**48**	37	57	92	86	33

Pada titik ini, 25 telah berada di posisi yang seharusnya, yaitu posisi ke-2, dan semua item di sisi kirinya lebih kecil dari atau sama denagn 25, dan semua item disisi kanannya lebih besar dari atau sama dengan 25.

Sekarang proses dilanjutkan untuk mengurutkan dua subarray yang terbentuk, yaitu (12) dan (48 37 57 92 86 33) dengan menerapkan metode yang sama.

Algoritma di atas dapat diimplementasikan dengan prosedur seperti berikut ini :

```
Procedure Partition ( LB, UB : integer; var j : integer );
Var up, down, A, temp : integer;
Begin
        ( A adalah pivot tabel yang akan ditentukan posisinya )
        A := X[LB]
        Up := UB; down := LB;
        While ( down < up ) do
        Begin
                While { X[down] <= A) and ( down < UB ) do
                        Down:= down + 1;

                While (X[up] > A) do
                        Up:= Up + 1;

                If (down < Up) then
                Begin
                        (pertukaran X[up] dan X[down])
                        tenp := X[up];
                        X[up] := X[down];
                        X[down] := tenp;
                End;
        End;
        X[LB] := X[Up];
        X[Up] := A;
        J := Up;
End;
```

```
Procedure QuickSort ( LB, UB : integer);
Var j : integer;
Begin
        If (LB < UB) then
                Begin
                        Partition(LB,UB,j);
                        QuickSort(LB,j-1);
                        QuickSort(j+1,UB)
                End;
        End;
End;
```

Bentuk algoritma lain yang juga dapat diterapkan dalam metode Quick Sort adalah sebagai berikut :

- Definisikan 2 buah pointer I dan j. Lakukan inisialisasi I = 1 dan j = n (jumlah data),
- Bandingkan X[i] dan X [j],

  Jika X[i] < X [j], tidak perlu dilakukan pertukaran, dan nilai j dikurangi 1, lalu proses di atas diulangi,

  Jika X[i] > X [j], lakukan pertukaran,
- Setelah terjadi pertukaran untuk pertama kali, nilai i ditambah 1. Proses perbandingan dilanjutkan dan nilai i ditambah 1 hingga terjadi pertukaran berikutnya,
- Setelah terjadi pertukaran untuk kedua kali, nilai j dikurangi 1 kembali, demikian seterusnya.

Dengan kata lain, dilakukan proses "*burning the candle at both ends*" (membakar lilin pada kedua ujungnya). Berikut ini diberikan sebuah contoh eksekusi algoritma di atas. (Nilai X[i] dan X [j] dicetak tebal).

- Kondisi awal (i = 1, j = n = 16, X[i] = 503, X [j] = 703 :

503	87	512	61	908	170	897	275
653	426	154	509	612	677	765	**703**

- Kurangi nilai j hingga j = 11, X[1] > X[11], atau 503 > 154, terjadi pertukaran ke-1; i = 1, j = 11.

154	87	512	61	908	170	897	275
653	426	**503**	509	612	677	765	703

- Tambah nilai i hingga i = 3, X[3] > X[10], atau 512 > 503, terjadi pertukaran ke-2; i = 3, j = 11.

154	87	**503**	61	908	170	897	275
653	426	**512**	509	612	677	765	703

- Kurangi nilai j hingga j = 10, X[3] > X[10], atau 503 > 426, terjadi pertukaran ke-3; i = 3, j = 10.

154	87	**426**	61	908	170	897	275
653	**503**	512	509	612	677	765	703

- Tambah nilai i hingga i = 5, X[5] > X[10], atau 908 > 503, terjadi pertukaran ke-4; i = 5, j = 10.

154	87	426	61	**503**	170	897	275
653	**908**	512	509	612	677	765	703

- Kurangi nilai j hingga j = 8, X[5] > X[8], atau 503 > 275, terjadi pertukaran ke-5; i = 5, j = 8.

154	87	426	61	**275**	170	897	**503**
653	908	512	509	612	677	765	703

- Tambah nilai i hingga i = 7, X[7] > X[8], atau 897 > 503, terjadi pertukaran ke-6; i = 7, j = 8.

154	87	426	61	275	170	**503**	**897**
653	908	512	509	612	677	765	703

Perhatikan bahwa semua perbandingan dalam contoh di atas melibatkan item data 503. Secara umum, tiap perbandingan akan melibatkan nilai awal X[1], karena data tersebut terus mengalami pertukaran posisi tiap kali arah proses diganti (nilai i ditambah atau nilai j dikurangi). Pada saat i = j , item data X[1] ini telah mencapai posisi yang seharusnya karena terlihat bahwa tidak ada lagi data yang lebih besar yang terletak disisi kirinya atau data yang lebih kecil disisi kanannya.

File awal telah mengalami pemilihan (*partitioned*) sedemikian rupa sehingga masalah pengurutan awal telah berubah menjadi 2 buah masalah baru yang lebih sederhana, yaitu mengurutkan X[1] s/d X[i-1] dan X[i+1] s/d X[n]. Algoritma diterapkan untuk mengurutkan kedua sub file ini.

Di bawah ini dituliskan hasil lengkap tiap iterasi untuk contoh di atas. Tanda kurung siku menunjukan subfile-subfile yang masih akan diurutkan dan yang dicetak tebal menunjukan item data telah menempati posisi seharusnya. Dalam progra, subfile ini dapat

direpresentasikan dengan dua variable LB(*lower bound*) dan UB(*Upper bound*), yang menunjukkan batas dari subfile yang sedang diproses. Disamping itu juga terdapat sebuah *stack* yang berisi nilai variable LB dan UB untuk subfile-subfile lain yang menunggu untuk diproses.

- LB = 1, UB = 16, Stack =[∅]

  [503 87 512 61 908 170 897 275
  653 426 154 509 612 677 765 703]

- LB = 1, UB = 6, Stack =[(8,16)]

  [154 87 426 61 275 170] **503**
  [897 653 908 512 509 612 677 765 703]

- LB = 1, UB = 2, Stack =[(4,6),(8,16)]

  [61 87] **154** [426 275 170] **503**
  [897 653 908 512 509 612 677 765 703]

- LB = 4, UB = 6, Stack =[(8,16)]

  61 87 154 [426 275 170] **503**
  [897 653 908 512 509 612 677 765 703]

- LB = 4, UB = 5, Stack =[(8,16)]

  61 87 154 [170 275] **426 503**
  [897 653 908 512 509 612 677 765 703]

- LB = 8, UB = 16, Stack =[∅]

  61 87 154 170 275 426 503
  [897 653 908 512 509 612 677 765 703]

- LB = 8, UB = 14, Stack =[∅]

  61 87 154 170 275 426 503
  [703 653 765 512 509 612 677] **897 908**

- LB = 8, UB = 12, Stack =[∅]

  61 87 154 170 275 426 503
  [653 512 509 612 677] **703 765 897 908**

- LB = 8, UB = 11, Stack =[∅]

  61 87 154 170 275 426 503
  [653 512 509 612] **677 703 765 897 908**

- LB = 9, UB = 11, Stack =[∅]

  61 87 154 170 275 426 503 509
  [653 512 612] **677 703 765 897 908**

- LB = 9, UB = 10, Stack =[∅]

  61 87 154 170 275 426 503 509
  [512 612] **653 677 703 765 897 908**

- LB = -, UB = -, Stack =[∅]

  61 87 154 170 275 426 503 509
  512 612 653 677 703 765 897 908

Rata-rata jumlah perbandingan untuk metode quick sort adalah O(n log n). Quick sort menunjukan performansi terbaik kasus array / file yang terurut sama sekali. Kasus terburuk untuk metode ini terjadi bila tabel terurut, karena jumlah perbandingan menjadi berorde $O(n^2)$. Hal sebaliknya berlaku untuk metode bubble sort (kasus terbaiknya adalah data terurut). Dalam kasus seperti ini, metode quick sort juga tidak lebih baik dibandingkan dengan selection sort.

Karena berorde O(n log n) dan overhead-nya yang rendah, metode quick sort menjadi metode tercepat yang sering digunakan dalam praktek.

4.4. IN TERNAL SORTING : SORTING BY SELECTION

Dalam metode selection sort, item data dipilih satu persatu sesuai dengan urutannya dan ditempatkan ke posisinya yang benar. Item-item data input tersebut mungkin perlu melalui tahap pemrosesan awal (preprocessing) terlebih dahulu agar pemilihan secara terurut (ordered selection) dapat dilakukan. Semua jenis selection sort secara konseptual dapat dituliskan dalam bentuk algoritma umum berikut, dimana digunakan sebuah *descending priority queue* (DPQ). Perlu diingat bahwa operasi PQ Insert menambahkan elemen baru ke dalam priority queue dan operasi PQ MaxDelete mengambil elemen dengan nilai terbesar dari dalam priority queue.

```
Set DPQ menjadi sebuah descending priority queue kosong;
(lakukan preprocessing terhadap item-item dalam array input dengan cara menginsertnya ke
dalam priority queue)
for I = 1 to N do
        PQ Insert (DPQ,X[ i ] )
(pilih tiap elemen satu per satu sesuai urutannya)
for I = N down to 1 do
        X[ i ]  := PQ MaxDelete (DPQ)
```

Algoritma ini disebut **General Selection Sort**. Perhatikan bahwa tahap selection dilakukan dengan serangkaian penghapusan (*deletion*) item dari dalam priority queue, meskipun sebenarnya yang dibutuhkan adalah penelusuran queue dengan urutan menurun (*descending order*).

Berikut ini akan ditinjau beberapa metode selection sort. Ada dua hal yang membedakan satu metode dengan metode yang lain. Pertama adalah struktur data yang digunakan untuk mengimplementasikan priority queue. Kedua adalah metode yang digunakan untuk mengimplementasikan algoritma umum di atas.

4.4.1. METODE STRAIGHT SELECTION SORT

Metode straight selection sort mengimplementasikan sdescending priority queue sebagai sebuah array tak terurut. Array input X digunakan untuk mengelola priority queue, sehingga tidak mengakibatkan adanya tambahan kebutuhan *space/memori* . Array input X merupakan sebuah array tak terurut sehingga direpresentasikan descending priority queue dengan

elemen. Karena bentuk inputnya sudah seperti yang dikehendaki, tidak perlu lagi dilakukan tahap *preprocessing*.

Karena itu, metode ini seluruhnya hanya terdiri dari tahap selection dimana secara berulang-ulang nilai data terbesar / ter kecil (variable *max* atau *min*) dianatar data yang tersisa (belum terurut) ditempatkan keposisi yang seharusnya. Untuk melakukan hal ini, *max/min* dipertukarkan letaknya dengan elemen X[i], dimana i adalah nomor iterasi. Priority queue yang pada awalnya berisi n elemen mengalami pengurangan elemen satu demi satu pada setiap iterasi. Setelah (n-1) kala selection, keseluruhan array menjadi terurut.

Jadi langkah-langkah yang dilakukan dalam metoda ini adalah :

- Pilih item data dengan nilai data terkecil (*minimum selection sort*) atau terbesar (*maximim selection sort*),
- Pertukarkan posisinya dengan item data pertama X[i],
- Ulangi operasi ini untuk (N-1) item data yang tersisa, kemudian untuk (N-2) item data yang tersisa, hingga tinggal 1 item data yang tersisa, yaitu item data ternesar/terkecil.

Hasil akhir dari minimum selection sort adalah data terurut menaik (ascending), sedangkan hasil akhir dari maximum selection sort adalah data terurut menurun (descending). Dua contoh kasus untuk masing-masing jenis selection sort tersebut diberikan di bawah ini :

4.4.1.1. MINIMUM SELECTION SORT

Awal	25	57	48	37	12	92	86	33
Iterasi 1	**12**	57	48	37	25	92	86	33
Iterasi 2	**12**	**25**	48	37	57	92	86	33
Iterasi 3	**12**	**25**	**33**	37	57	92	86	48
Iterasi 4	**12**	**25**	**33**	**37**	57	92	86	48
Iterasi 5	**12**	**25**	**33**	**37**	**48**	92	86	57
Iterasi 6	**12**	**25**	**33**	**37**	**48**	**57**	86	92
Iterasi 7	**12**	**25**	**33**	**37**	**48**	**57**	**86**	**92**

Awal	44	55	12	42	94	18	6	67
Iterasi 1	**6**	55	12	42	94	18	44	67
Iterasi 2	**6**	**12**	55	42	94	18	44	67
Iterasi 3	**6**	**12**	**18**	42	94	55	44	67
Iterasi 4	**6**	**12**	**18**	**42**	94	55	44	67
Iterasi 5	**6**	**12**	**18**	**42**	**44**	55	94	67
Iterasi 6	**6**	**12**	**18**	**42**	**44**	**55**	94	67
Iterasi 7	**6**	**12**	**18**	**42**	**44**	**55**	**67**	**94**

4.4.1.2. MAXIMUM SELECTION SORT

Awal	25	57	48	37	12	92	86	33
Iterasi 1	**92**	57	48	37	12	25	86	33
Iterasi 2	**92**	**86**	48	37	12	25	57	33
Iterasi 3	**92**	**86**	**57**	37	12	25	48	33
Iterasi 4	**92**	**86**	**57**	**48**	12	25	37	33
Iterasi 5	**92**	**86**	**57**	**48**	**37**	25	12	33
Iterasi 6	**92**	**86**	**57**	**48**	**37**	**33**	12	25
Iterasi 7	**92**	**86**	**57**	**48**	**37**	**33**	**25**	**12**

Awal	44	55	12	42	94	18	6	67
Iterasi 1	**94**	55	12	42	44	18	6	67

Iterasi 2	**94**	**67**	12	42	44	18	6	55
Iterasi 3	**94**	**67**	**55**	42	44	18	6	12
Iterasi 4	**94**	**67**	**55**	**44**	42	18	6	12
Iterasi 5	**94**	**67**	**55**	**44**	**42**	18	6	12
Iterasi 6	**94**	**67**	**55**	**44**	**42**	**18**	6	12
Iterasi 7	**94**	**67**	**55**	**44**	**42**	**18**	**12**	**6**

Berikut ini dituliskan dua versi kode program untuk prosedur minimum straight selection sort. Perbedaan diantara keduanya hanya sedikit, yaitu pada titik awal penelusuran array untuk penempatan item data terkecil / terbesar.

```
Procedure MinSelection ( Var X : ArrayType; N : integer );
Var i, j, idx, max : integer ;
Begin
        For i := N down to 2 do
        Begin
                Max := x[1];
                Idx :=1;
                For j := 2 to i do
                        If (X[j] > max ) then
                        Begin
                                Max := x[j];
                                Idx :=j;
                        End;
                        x[idx] : = x[i]
                        x[i] := max;
        end;
end;
```

```
Procedure MinSelection ( Var X : ArrayType; N : integer );
Var i, j, idx, max : integer ;
Begin
        For i := 1  to N-1 do
        Begin
                Min := x[i];
                Idx :=i;
                For j := ( i+1)  to N do
                        If (X[j] < min ) then
                        Begin
                                Min := x[j];
                                Idx :=j;
                        End;
                        x[idx] : = x[i]
                        x[i] := min;
        end;
end;
```

Dalam straight selection sort, jumlah perbandingan yang dilakukan adalah (n^2–n)/2. Jadi jumlah perbandingan berorde O(n^2). Jumlah pertukaran yang dilakukan selalu n-1 (kecuali jika dilakukan pengecekan terlebih dahulu untuk menghindari sebuah item data dipertukarkan dengan dirinya sendiri). Walaupun ordenya O(n^2), metode ini lebih cepat dibandingkan bubble sort. Tidak ada perubahan performansi apabila file input telah berurut atau sama sekali belum terurut. Penggunaan metoda ini dianjurkan untuk jumlah data yang kecil

4.5. INTERNAL SORT BY INSERTION

Metode insertion sort ialah teknik yang mengurutkan sekumpulan record dengan cara menyisipkan record-record ke dalam sebuah file yang telah terurut.

4.5.1. METODE STRAIGHT INSERTION SORT

Metode straight insertion / simple sort dapat dipandang sebagai sebuah generasi *general selection sort* dimana priority queue-nya diimplementasikan sebagai sebuah array terurut. Yang dibutuhkan hanyalah tahap *preprocessing* untuk meng-insert item-item data ke dalam priority queue. Setelah item-item tersebut selesai di insert, semua telah terurut sehingga tidak diperlukan tahap *selection*.

Jika file awalnya telah terurut, hanya dibutuhkan satu kali pembandingan pada tiap iterasi, sehingga ordernya menjadi O(n). Jika file awalnya terurut terbalik (*reverse order*), orde metode ini menjadi $O(n^2)$, karena jumlah pembandingan yang dilakukan adalah :

$$(n-1)+ (n-2)+ (n-3)+ \ldots + 3+ 2 + 1 = (n^2 - n)/2$$

Meskipun ordenya sama, staright insertion sort umumnya lebih baik dibandingkan bubble sort. Semakin terurut file data awalnya, semakin efisien pula metoda ini bekerja. Rata-rata jumlah pembandingan dalam metode ini adalah (n^2). Space tambahan hanya dibutuhkan untuk menyimpan sebuah variable sementara (*template*).

Berikut ini diberikan dua buah contoh pengurutan dengan straight insertion sort. Pada gambar dibawah ini dituliskan kode program dalam bahasa Pascal untuk metode ini.

Awal	25	57	48	37	12	92	86	33
Iterasi 1	**25**	57	48	37	12	92	86	33
Iterasi 2	**25**	**48**	57	37	12	92	86	33
Iterasi 3	**25**	**37**	**48**	**57**	12	92	86	33
Iterasi 4	**12**	**25**	**37**	**48**	**57**	92	86	33
Iterasi 5	**12**	**25**	**37**	**48**	**57**	**92**	86	33
Iterasi 6	**12**	**25**	**37**	**48**	**57**	**86**	**92**	33
Iterasi 7	**12**	**25**	**33**	**37**	**48**	**57**	**86**	**92**

Awal	44	55	12	42	94	18	6	67
Iterasi 1	**44**	**55**	12	42	94	18	6	67
Iterasi 2	**12**	**44**	**55**	42	94	18	6	67
Iterasi 3	**12**	**44**	**42**	**55**	94	18	6	67
Iterasi 4	**12**	**44**	**42**	**55**	**94**	18	6	67
Iterasi 5	**12**	**18**	**44**	**42**	**55**	**94**	6	67
Iterasi 6	**6**	**12**	**18**	**44**	**42**	**55**	**94**	67
Iterasi 7	**6**	**12**	**18**	**44**	**42**	**55**	**67**	**94**

```
Procedure Insertion (var X:ArrayType; N:integer);
Var     k, i, y    : integer;
        Found   : boolean;
Begin
        (X[1] merupakan array yang terurut dengan 1 item.
         Di akhir tiap iterasi, item X[1]-X[k] telah terurut.)
        For k := 2 to N do
        Begin
                Y := X[k];
                ( Semua data yang lebih besar dari y di geser Satu
                  posisi ke sebelah kanan )
                i := k – 1;
                found := false;
                while ( i >= 1) and (not found) do
                        if (y < X[i]) then
                        begin
                                X[i+1] := X[1];
                                i :=  i –1;
                        end
                        else found := true;
                ( insert X[k] ke posisi yang tepat )
                X(i+1) := y;
        End;
End;
```

4.5.2. SHELL SORT (DIMINISHING INCREMENT SORT)

Metode shell sort dibuat oleh Donald L. Shell (1985). Metode ini membagi – bagikan file data awal menjadi beberapa subfile. Subfile – subfile ini berisi setiap item data kelipatan ke – k dalam file awalnya. Nilai k ini disebut increment.

Sebagai contoh, jika k = 5, maka subfile yang pertama diproses adalah subfile yang berisi item data X[1], X[6], X[11],… Dengan cara seperti ini akan terdapat 5 buah yang masing – masing berisi 1/ 5 data dari file awal, yaitu :

Subfile 1	X[1]	X[6]	X[11] dst
Subfile 2	X[2]	X[7]	X[12] dst
Subfile 3	X[3]	X[8]	X[13] dst
Subfile 4	X[4]	X[9]	X[14] dst
Subfile 5	X[5]	X[10]	X[15] dst

Secara umum, jika digunakan nilai increment k (terdapat k buah subfile), maak elemen data ke – I dari subfile ke- j adalah X[I-1]x k + j)

Setelah k buah subfile ini selesai diurutkan (biasanya dengan menggunakan metode straigt insertion sort), dipilih sebuah nilai baru yang lebih kecil untuk increment k dan file kembali kembali dibagi ke dalam kumpulan subfile – subfile baru berdasarkan nilai k ini. Masing – masing subfile yang berukuran lebih besar ini (karena nilai k lebih kecil) diurutkan, dan proses ini diulangi kembali dengan nilai k yang lebih kecil.

Pada akhirnya, nilai increment k ini diset sama dengan satu, sehingga yang diproses adalah 1 buah subfile yang berisi keseluruhasn data yang ada difile awal. Urutan nilai increment yang mengecil ini telah ditetapkan pada awal proses sorting (diminish beraryi mengurangi)

Sebagi contoh, jika file awal berisi item – item data beriktu :

25 57 48 37 12 92 86 33

dan urutan nilai increment k yang dipilih adalah (5,3,1), maka pada tiap iterasi akan diurutkan subfile – subfile berikut :

Iterasi 1 (Increment = 5)

(X[1[, X[6]) (X[2], X[7])

(X[3], X[8]) (X[4])

(X[5])

iterasi 2 (increment k = 3)

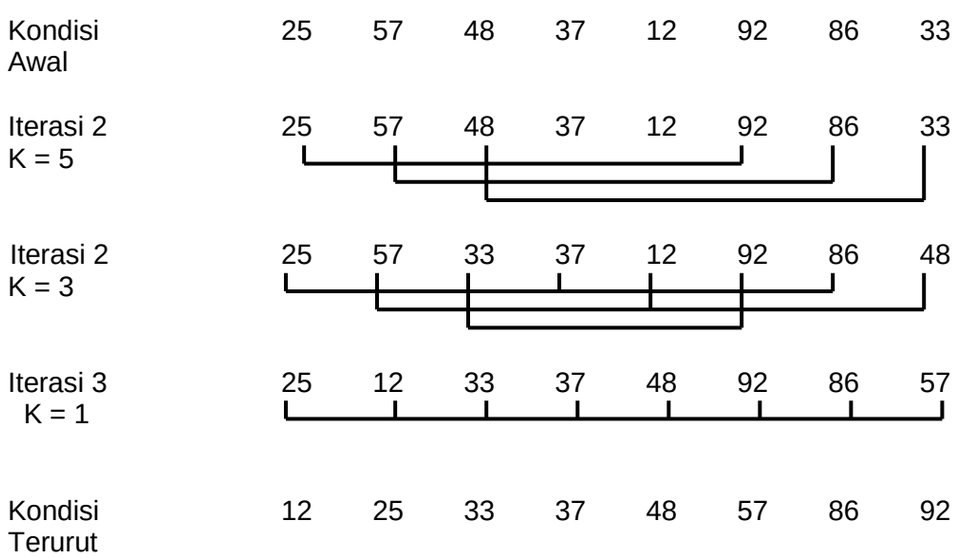
(X[1], X[4], [7]) (X[2], X[5], X[8])

(X{3],X[6]

iterasi 3 (increment k = 1)

(X[1], X[2], X[3], X[4], X[5], X[6], X[7], X[8])

Tahapan proses pengurutannya ditunjukkan beriktu ini. Item data yang berada dalam subfile yang sama dihubunhkan dengan tanda garis. Tiap subfile tersebut diurutkan dengan menggunakan metode straight insertion sort.

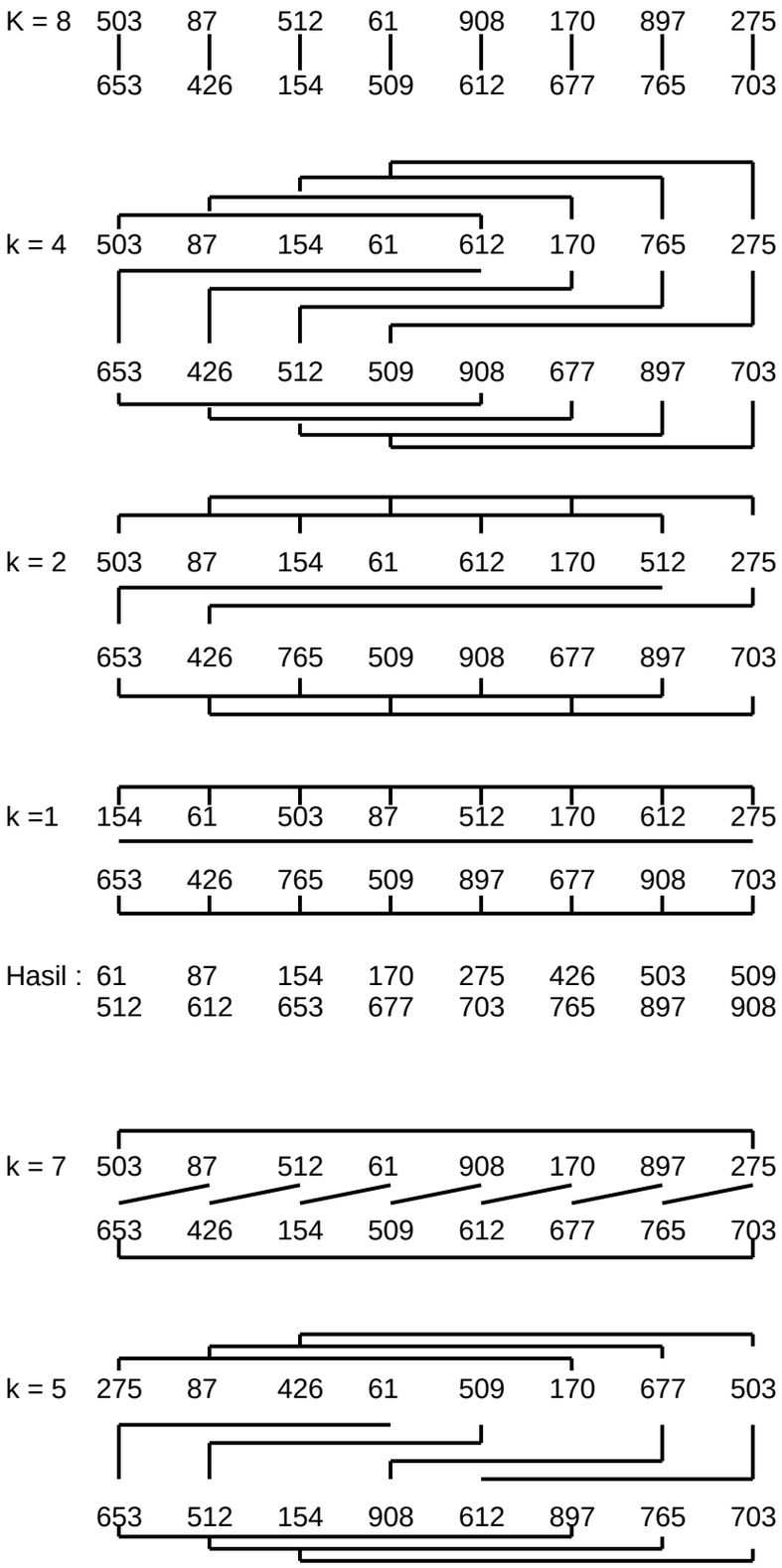


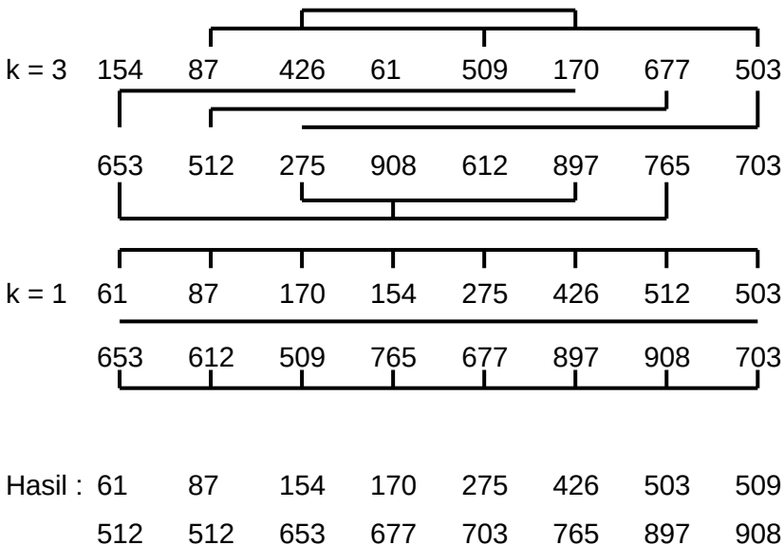
Ide di belakang metode shell sort ini cukup sederhana. Telah disebutkan seblumnay bahwa starigt insertion sort sangat efisein untuk file data yang hampir terurut. Juga penting untuk disadari bahwa apabila ukuran file kecil, maka metode sort berorde) (n2) sering lebih efisien dibandingkan metode yang berorde 0 (n log n). Karena nilai increment pertama yang digunakan oleh shell sort cukup besar, maka ukuran tiap subfile menjadi kecil, sehingga proses staright insertion terhadap subfile – subfile ini menjadi cepat. Tiap pengurutan subfile ini membuat keseluruhan file menjadi semakin terurut.

Karena itu meskipun iterasi – iterasi berikutnya menggunakan nilai increment yang lebih kecil sehingga menyebabkan subfile semakain besar, namun subfile – subfile tersebut sudah berada dalam kondisi hampir terurut sebagi hasil dari iterasi sebelumnya. Insertion sort terhadap subfile – subfile ini juga akan berlangsung cukup efisien.

Orde metode shell sort adalah 0 (n (log N)2) jika digunakan urutan nilai increment yang tepat. Knuth merekomendasikan pemilihan nilai increment sebagai berikut :
Definisikan fungsi f secara rekursif sehingga h910 =1 dan h(I +1) = 3xh(i0 +. Misalkan x adalh bilangan bulat terkecil sedemikian rupa sehingga h(x) .n. Set nilai Numeric (jumlah increment) menjadi x –2 dan set Inc[1] menjadi h(Numeric –1 +1) untuk I mulai dari s.d. NumInc.

Berikut ini akan diberikan contoh kedua tangan n = 16 dan dua alternatif urutan k , yaitu (8,4,2,1) dan (7,5,3,1).





Hasil : 61 87 154 170 275 426 503 509
512 512 653 677 703 765 897 908

Di bawah ini dituliskan kode program dalam bahsa Pascal untuk metode shell sort.

```
Type IncTab = record
          NumInc  : 1 .. NumElm
          Incrm      : array [ 1..numElm] of integer ;
       End;
Procedure Shell ( var X : ArrayType; N:integer; IncTab) ;
Var  j, span       : integer;
      Incr, y,  k   : integer;
      Found        : boolean;

Begin
     For incr := 1 to Inc. NumINc do
     Begin
        {span adlah nilai increment yang current}
        span : = Inc.Incrm [   incr];
        for j  : =          spant+1 to N do
        begin
        {insert elemen xx[j] ke posisinya yang tepat dalam subfilenya}
        y : = X[ j ];
        k  := j – span;
        found  : = false;
        while ( k > =1) and ( not found ) do
                if ( y < X[k]) then
                begin
                      x[k+span] : = X[k];
                      K : = k – spasi;
                     End
                Else found found : = true ;
        X[ k+span] : = y:
     End;
End;
End;
```

4.6. EXTERNAL SORTING

Metode – metode internal sorting yang diuraikan di atas tidak dapat diterapkan apabila junlah item data yang akan diurutkan tidak dapat ditampung dalam memori utama komputer (sehingga harus disimpan dalam media penyimpanan sekunder, seperti disk). Dalam

kasus seperti ini, data disimpan dalam bentik sebuah file, dimana pada setiap saat hanya satu item data yang dapat diakses. Hal ini menjadi suatu batasan jika dibandingkan terhadap struktur array. Karena itu harus digunakan teknik sorting yang berbeda. Salah satu diantaranya yang terpenting adalah sorting by merging.

4.6.1. METODE STRAIGHT MERGING SORT

Merging ialah proses mengabungkan dua atau lebih file yang terurut ke dalam file ketiga yang dapat diakses secara berulang – ulang. Merging merupakan sebuah operasi yang jauh lebih sederhana dibandingkan sorting.

Contoh sebuah prosedur yang menerima masukkan dua buah array terurut A dan B dengan jumlah elemen data an dan bn, dan mengabungkan keduanya ke dalam array ketiga C yang akan berisi C_n elemen data.

```
Procedure MergeArray ( A, B : ArrayType; Var C : ArrayType;
                                An, bn : integer; Var cn : integer );
{ Merge/gabungkan array A dan array B ke dalam array C }
Var Aidx, Bidx, Cidx : integer;
Begin
If ( an + bn > NumElm ) then
    Error ( ‘ Data yang akan di-merge terlalu banyak  ‘ );
Else begin
       Cn := an + bn;
      Aidx := 1; Bidx := 1; Cidx :=1;
      While ( Aidx <= an ) and ( Bidx <+ bn ) do
        begin
                if ( A[Aidx] < B[idx] then
                 begin
                        C[Cidx] := A[Aidx];
                         Aidx := Aidx + 1;
                  Endelse
                  Begin
                       C[Cidx] := B[Bidx];
                       Bidx := Bidx + 1;
                   End;
                    Cidx := Cidx + 1;
          End;
          { Salin elemen data yang tersisa di array A dan B }
           While ( Aidx <= an ) do
             Begin
                     C[Cidx] := A[Aidx];
                      Cidx := Cidx + 1;
                       Aidx := Aidx + 1;
            End;
             While ( Bidx <= bn ) do
             Begin
                      C[Cidx] := B[Bidx];
                       Cidx := Cidx + 1;
                        Bidx := Bidx + 1;
             End;
       End;
  End;
```

Pada prosedur di atas, dua buah array / tabel terurut digabungkan ke dalam sebuah array yang baru. Peoses ini dilakukan dengan cara memilih record dengan nilai key paling kecil

dalam kedua array tersebut secara berulang -–ulang dan menempatkannya ke dalam array yang baru. Sebuah contoh dengan 5 item deberikan berikut ini.

Array 1		11	23	42	
Array 2			9	25	
Array 3					
Array 1			11	23	42
Array 2			25		
Array 3			9		
Array 1	23	42			
Array 2	25				
Array 3	**9**	**11**			
Array 1	42				
Array 2	25				
Array 3	**9**	**11**	**23**		
Array 1	42				
Array 2					
Array 3	**9**	**11**	**23**	**25**	
Array 1					
Array 2					
Array 3	**9**	**11**	**23**	**25**	**42**

Kita dapat menggunakan teknk ini untuk mengurutkan sebuah file dengan cara berikut. Pertama, kita membagi file yang akan diurutkan menjadi n buah subfile yang hanya berisi 1 data. Kemudian, pasangan subfile yang bertetangga (*adjacent*) digabungkan untuk membentuk subfile baru yang berukuran lebih besar. Jadi, kita memperoleh kira-kira n/2 buah subfile yang masing-masing berisi 2 data. Proses ini diulangi hingga akhirnya diperoleh 1 buah *file* yang berukuran n data. Gambar 1 menunjukkan bagaimana proses ini dilaksanakan pada sebuah file contoh. Setiap subfile ditandai dengan kurung siku.

Berikut ini akan diberikan sebuah prosedur untuk mengimplementasikan teknik **Straight Merge Sort** yang telah dijelaskan di atas. Dibutuhkan sebuah array bantu *Aux* berukuran *NumElm* untuk menyimpan hasil penggabungan dua buah subarray dari X. Variabel *size* digunakan untuk mengontrol ukuran dari subarray yang sedang digabungkan. Karena kedua file yang sedang digabungkan merupakan subarray dari X, maka dibutuhkan batas bawah (*lower bound*) dan batas atas (*upper bound*) untuk menunjukkan subfile-subfile dari X yang sedang diproses. LB1 dan UB1 merupakan batas bawah dan batas atas untuk file pertama; LB2 dan UB2 merupakan batas bawah dan batas atas untuk file kedua. Variabel I dan j digunakan untuk mengacu ke elemen data pada file sumber (*source file*) yang sedang diproses dan k mengacu ke file tujuan (*destination file*). Prosedurnya dituliskan dengan lengkap di bawah ini.

```
Procedure MergeSort (var X : ArrayType; N : integer);
Var     Aux : ArrayType;
        LB1, LB2, UB1, UB2 : integer;
        i, j, k, size : integer;
Begin
        size := 1; {merge file-file berukuran 1 data}
        while (size < N) do
        begin
                {inisialisasi batas bawah dari file pertama}
                LB1 := 1;
                k := 1;
                while (LB1 + size <= N) do
                {cek apakah ada 2 file yang akan di-merge}
                begin
                        LB2 := LB1 + size;
                        UB1 := LB2 – 1;
                        If (LB2 + size – 1 > N)
                                Then UB2 := N
                        Else    UB2 := LB2 + size – 1;
                End;
                i := LB1;
                j := LB2;
                while (I <= UB1) and (j <= UB2) do
                begin
                        if (X[i] <= X[j]) then
                        begin
                                Aux[k] := X[i];
                                i := i + 1;
                        end else
                        begin
                                Aux[k] := X[j];
                                j := j + 1;
                        end;
                        k := k + 1;
`               end;
                while (i <= UB1) do
                begin
                        aux[k] := X[i];
                        i := i + 1;
                        k := k + 1;
                end;
                while (j <= UB2) do
                begin
                        aux[k] := X[j];
                        j := j + 1;
                        k := k + 1;
                end;
                LB1 := UB2 + 1;
                {salin isi file-file yang masih tersisa}
                i := LB1;
                while (k <= N) do
                begin
                        aux[k] := X[i];
                        k := k + 1;
                        i := i + 1;
                end;
                for k := 1 to N do X[k] := Aux[k];
                size := size * 2;
        end;
end;
```

Di bawah ini diberikan dua buah contoh proses straight merging untuk file data berukuran 8 dan 10.

Awal	[25]	[57]	[48]	[37]	[12]	[92]	[86]	[33]
Pass 1	[25	57]	[37	48]	[12	92]	[33	86]
Pass 2	[25	37	48	57]	[12	33	86	92]
Pass 3	[12	25	33	37	48	57	86	92]

Awal	[42]	[23]	[74]	[11]	[65]	[58]	[94]	[36]	[99]	[87]
Pass 1	[23	42]	[11	74]	[58	65]	[36	94]	[87	99]
Pass 2	[11	23	42	74]	[36	58	65	94]	[87	99]
Pass 3	[11	23	36	42	58	65	74	94]	[87	99]
Pass 4	[11	23	36	42	58	65	74	87	94	99]

Metode di atas disebut *2-way merging*. Cara ini dapat digeneralisasi untuk menggabungkan k buah tabel terurut ke dalam satu array terurut. Teknik merging sedemikian disebut *multiple merging* atau *k-way merging*.

Metode sorting ini cukup efisien. Karena jumlah iterasi yang dibutuhkan adalah $[\log_2 n]$ maka jumlah perbandingan yang dilakukan berorde $O(n \log_2 n)$. Orde ini berlaku untuk kasus terburuk dan juga kasus rata-rata. Salah satu kelemahan utaman metode ini adalah area data sekunder yang dibutuhkan cukup besar. Merge sort membutuhkan tambahan space untuk array bantu sebesar O(n), sedangkan quicksort hanya membutuhkan tambahan space sebesar O(log n) untuk stack.

Salah satu modifikasi yang dapat dilakukan terhadap metode di atas adalah dengan menggunakan *linked allocation* untu menggantikan alokasi sekuensial. Dengan menambahkan sebuah filed pointer tunggal ke dalam tiap record, kebutuhan akan array bantu *Aux* dapat ditiadakan. Hal ini dilakukan dengan menghubungkan secara eksplisit setiap subfile input dan output. Modifikasi ini dapat dilakukan baik terhadap straight merge maupun natural merge.

4.6.2. Metode Natural Merging Sort

Dalam metode straight merging tidak diperoleh keuntungan apa-apa bila data pada awalnya telah terurut sebagian (*partially sorted*). Ukuran dari semua subfile yang telah di-merge pada iterasi/pass ke-k adalah kurang dari atau sama dengan 2^k, tanpa melihat apakah subfile lainnya telah terurut dan dapat digabungkan juga. Pada kenyataannya, dua buah subfile sebarang dengan panjang m dan n dapat digabungkan secara langsung ke dalam sebuah file tunggal dengan jumlah item m + n. Metode mergesort yang pada setiap saat menggabungkan dua buah subfile terpanjang yang mungkin disebut **natural merging sort**.

Jadi, jika pada straight merge,s emua subfile berukuran sama (kecuali mungkin untuk subfile terakhir), maka pada natural merge kita memanfaatkan beberapa keterurutan yang telah ada dalam file awal dan pemilahannya dilakukan berdasarkan subfile terpanjang dengan urutan data yang menaik (*increasing*). Dengan kata lain, kita memanfaatkan derajat keterurutan yang ada dalam array awal untuk melakukan *2-way merge sort*. Di bawah ini diberikan dua buah contoh proses natural merging untuk 10 item dan 20 item.

```
Awal    [70]  [10  17  75]  [42]  [35]  [29  53  65]  [48]
Pass 1  [10   17  70  75]   [35   42]   [29  48  53   65]
Pass 2  [10   17  35  42    70    75]   [29  48  53   65]
Pass 3  10    17  29  35    42    48    53  65  70   75
```

Kondisi awal :

```
[17  31]  [5   59]  [13  41  43  67]  [11  23  29  47]
[3   7    71]  [2   19  57]  [37  61]
```

Pass 1 :

```
[5   17  31  59]  [11  13  23  29  41  43  47  67]
[2   3   7   19   57  71]  [37  61]
```

Pass 2 :

```
[5   11  13  17  23  29  31  41  43  47  59  67]
[2   3   7   19  37  57  61  71]
```

Pass 3 :

```
[2   3   5   7   11  13  17  19  23  29  31  37
41   43  47  57  59  61  67  71]
```

Penjelasan mengenai dua algoritma *sorting by merging* lainnya, yaitu **Balanced Multiway Merging** dan **Polyphase Sort**, dapat dilihat pada [WIR76].

4.7. KESIMPULAN

[WIR76] memberikan hasil perbandingan performansi dari beberapa metode sorting dalam bentuk tabel – tabel berikut. Pada tabel 2, n adalah jumlah data, C menunjukan jumlah perbandingan (comparison), dan M menunjukan jumah pertukaran jumlah pertukaran (move).

Tabel 2. Perbandingan metode – metode sorting sederhana

METODE	MINIMUM	RATA - RATA	MAKSIMUM
Straight exchange (bubble sort)	C = (n2 – n) / 2 M = 0	C = (n2 – n) / 2 M = (n2 – n) x 0,75	C = (n2 – n) / 2 M = (n2 – n) x 1,5
Straight selection	C = (n2 – n) / 2 M = 3 (n – 1)	C = (n2 – n) / 2 M = n (1n n + 0,75)	C = (n2 – n) / 2 M = n2 / 4 + 3 (n – 1)
Straight insertion	C = n – 1 M = 2 (n – 1)	C = (n2 + n – 2) / 4 M = (n2 – 9n – 10 / 4	C = (n2 – n) / 2 – 1 M = (n2 + 3n – 4) / 2

Tabel 3 di bawah ini menunjukan waktu yang dibutuhkan (dalam milisecond) oleh program sorting dalam bahasa pascal yang dieksekusi pada komputer CDC 6400. Ketiga kolom tersebut menunjukan waktu yang digunakan untuk mengurutkan array yang telah terurut, array yang urutannya acak, dan array yang terurut terbalik. Sisi kiri kolom adalah waktu untuk mengurutkan 246 item, sedangkan sisi kanan adalah untuk 512 item.

Tabel 3. Perbandingan waktu eksekusi dari metode – metode sorting

METODE	Terurut (ordered)		Acak (random)		Terurut terbalik (inversely ord)	
Staright insertion	12	23	366	1444	704	2836
Straight selection	483	1907	509	1956	695	2675
Bubble sort	540	2165	1026	4054	1492	5931
Shaker sort	5	9	961	3642	1619	6520
Shell sort	58	116	127	349	157	492
Heap sort	116	253	110	241	104	226
Quick sort	31	69	60	146	37	79
Straight merging	99	234	102	242	99	232

Dari tabel di atas terlihat bahwa :

i. Metode bubble sort merupakan metode sorting paling terburuk dinamdingkan metode – metode lainnya. Metode shaker sort, yang merupakan versi perbaikannya, masih lebih buiruk dibandingkan straight insertion dan straighty selection sort.

ii. Metode quick sort mengnguguli metode heap sort dengan faktor 2 – 3. Quick sort mengurutkan array yang terburuk terbalik denan kecepatan yang sama dengan array terurut.

Pada tabel 3 di atas, metode sorting mengurutkan record yang hanya terdiri dari sebuah key saja. Pada kebnyataanya, sebuah record memiliki banak field yang lain. Tabel 4 berikut menunjukan bagaimana pengaruh pemanbahan field ke dalam record terhadap performansi waktu metode sorting. Total ukuran record tanpa field tambahan, sedangkan sisi kanan kolom untuk record dengan field tambahan; jumlah daya n = 256

Tabel 4. Perbandingan waktu eksekusi dari metode – metode sorting
Untuk record dengan field tambahan

METODE	Terrut (ordered)		Acak (random)		Terurut terbalik (inversely ord.)	
Straight insertion	12	46	366	1129	704	2150
Straight selection	489	547	509	607	695	1430
Bubble sort	540	610	1026	3212	1492	5599
Shaker sort	5	5	961	3071	1619	5757
Shell sort	58	186	127	373	157	435
Heap sort	116	264	110	246	104	227
Quick sort	31	55	60	137	37	75
Straight merging	99	196	102	195	99	187

Kesimpulan dari hasil perbandingan metode-metode sorting diatas adalah sebagai berikut :

i. Metode straight selestion sort menunjukan keunggulan yang signifikan sebagai metode sederhana (straight method) yang terbaik.

ii. Metode bubble sort merupakan metode terburuk. Versi “perbaikan “ – nya, yaitu metode shaker soer hanya sedkiti lebih baik dibandingkan bubble sort dalam kasus array terurut terbalik (inversely ordered).

iii. Metode quick sort metode yang tercepat dan metode internal sorting yang terbaik.

Baik straight selection sort maupun simple insertion sort lebih sfesien dibandingkan bubble sort. Selection sort membutuhkan lebih sedikit assigment dibandingkan insertion sort, namun lebih banyak perbandinan. Karena itu, selection sort direkomendasikan untuk file – file berukuran kecil dengan ukuran record besar (assigment mahal, tetapi perbandingan murah karena key sederhana). Untuk kasus sebaliknya, direkomendasikan insertion sort.

BAB VI

SEARCHING (Pencarian)

Pencarian Data yang sering juga disebut dengan ***table look-up*** atau ***storage and retrieval information*** adalah suatu proses untuk mengumpulkan sejumlah informasi di dalam memori komputer dan kemudian mencari kembali informasi yang diperlukan secepat mungkin.
Seringkali kita dihadapkan pada permasalahan dimana sebenarnya kita memerlukan sedikit informasi dari sekian banyak informasi yang tersedia, sehingga sering timbul pemikiran untuk menghapus informasi yang tidak diperlukan. Namun dikemudian hari kita berubah pikiran untuk mempertahankan informasi tersebut. Dan mengorganisasikannya sehingga proses pencarian dapat dilakukan dengan cepat dan tepat.

6.1. Istilah yang digunakan

Secara umum kita memakai anggapan sejumlah data yang tersimpan dalam memori komputer, dan persoalan yang timbul adalah mencari data yang kita inginkan. Dalam setiap rekaman (***record***) biasanya terdiri dari medan (***field***). Untuk membedakan satu record dengan record yang lain maka harus ada satu field yang menjadi kunci (***key field***). Kumpulan dari record yang kita miliki dinamakan dengan berkas (***file***) atau tabel.
Hubungan antara key field dengan record biasanya sederhana atau kompleks. Bentuk paling sederhana dari sebuah key field adalah yang terletak dalam record tersebut. Key ini biasanya dinamakan ***internal key*** atau ***embedded key***. Key yang lain ada yang tersimpan di dalam tabel di luar dari tabel utama. Key ini biasanya dinamakan dengan ***external key***.
Jika dalam sebuah record terdapat field yang dapat membedakan record tersebut secara ***unique*** (berbeda untuk setiap recordnya) maka filed tersebut dinamakan ***primary key***. Sedangkan field yang membantu membedakan setiap record (tidak dapat berdiri sendiri) maka field tersebut sering dinamakan ***secondary key***. Beberapa kunci ada yang disajikan secara ***single key***, dan beberapa yang lain disajikan secara ***multiple key***.
Algoritma Pencarian (***Searching Algorithm***) adalah algoritma yang menerima kunci bernilai K dan dengan langkah-langkah tertentu akan mencari record yang kuncinya bernilai K. Setelah proses dijalankan, kemungkinan jawabannya hanya ada dua yaitu berhasil ditemukan (***succesful***) dan tidak berhasil ditemukan (***unsuccesful***). Pencarian yang berhasil menemukan sering dinamakan ***retrieval***.
Untuk algoritma pencarian dan menambahkan data, dinamakan dengan ***search and insertion algorithm***. Metoda pencarian data dapat dikelompokan kedalam beberapa metoda. Kelompok metoda pertama dapat dikelompokan menjadi **pencarian internal** (***internal searching***) dan **pencarian eksternal** (***external searching***).
Pengelompokan kedua adalah dengan mengelompokan ke dalam **pencarian statis** (***static searching***) dan **pencarian dinamis** (***dynamic searching***). Cara pengelompokan yang

lain adalah berdasarkan kunci yang ada atau berdasar sifat digital (***digital properties***) dari kunci yang dimaksud. Menggunakan perbandingan dan distribusi.

6.2. Pencarian Berurutan (*sequential searching*)

6.2.1. Pencarian Pada Tabel yang belum Diurutkan

Metode yang paling sederhana dari sejumlah metode pencarian adalah metoda pencarian berurutan (***sequential searching***). Secara garis besar metoda ini dapat dijelaskan sbb:
Dari Vektor yang dicari, data yang dicari dibandingkan satu persatu sampai data tersebut ditemukan atau tidak ditemukan. Pada saat data ditemukan maka proses pencarian langsung berhenti. Dalam kondisi yang paling buruk proses ini melakukan pencarian sebanyak N kali (sejumlah data yang ada).

```
Procedure CARI_URUT( Var Ada   : boolean;    N,Posisi    : integer;
                         A     : Array;      Data        : integer);
Kamus Lokal
     I : integer;
Algortima :
     (*dianggap tidak ada data*)
     Ada := False;
     (*iterasi dimulai*)
     For I := 1 to N do
          If A[I] = data then
               (*Data ketemu*)
               Begin
                    Posisi := I;
                    Ada := True;
               End
     Endfor
     Ada = false
     If not Ada then
(*Data yang dicari tidak ketemu sisipkan elemen ke dalam vektor*)

          Begin
               Inc(N)
               A[N] := Data;
          End
```

6.2.2. Pencarian Pada Tabel yang Sudah Diurutkan

Secara singkat pencarian pada tabel yang sudah diurutkan adalah :
Dari elemen pertama dibandingkan dengan elemen yang dicari. Jika elemen yang dibandingkan lebih kecil dari elemen yang di cari maka diteruskan pada elemen selanjutnya. Jika elemen yang dibandingkan lebih besar, maka dapat dipastikan elemen yang dicari tidak dapat ditemukan. Jika ditemukan, maka proses pencarian selesai.

6.2.2. Pencarian Biner / Binary Search

Setelah vektor yang dicari diurutkan, maka vektor tersebut dibagi dua sub vektor dengan jumlah data yang sama (Jika ganjil maka diambil dari hasil N div 2). Kemudian data yang

dicari dibandingkan dengan elemen data yang terakhir dari data sub vektor yang pertama. Apabila data yang dicari lebih besar dari elemen akhir dalam sub vektor sebelah kiri, maka data yang dicari kemungkinan ada pada sub vektor sebelah kanan.

Jika data yang dicari lebih kecil dari elemen akhir dalam sub vektor sebelah kiri , maka data yang dicari kemungkinan ada pada sub vektor sebelah kiri. Proses selanjutnya adalah kembali ke awal dengan menganggap sub vektor yang diperkirakan berisi data yang dicari dianggap vektor yang akan dibagi dua. Untuk lebih jelasnya perhatikan gambar di bawah ini

2	8	12	37	43	68	72	77	82	94

Vektor di atas dibagi menjadi dua sub vektor seperti di bawah ini :

2	8	12	37	43

Sub vektor 1

68	72	77	82	94

Sub Vektor 2

Misalnya data yang dicari adalah 12, maka data ada pada sub vektor1. Setelah itu kemudian sub vektor1 dibagi menjadi 2 bagian seperti di bawah ini :

2	8

Sub vektor 1

12	37	43

Sub Vektor 2

Karena data yang dicari ada pada Sub Vektor2, maka data ada pada sub vektor2 kemudian dibagi menjadi 2 bagian kembali seperti di bawah ini :

12

Sub vektor 1

37	43

Sub Vektor 2

Karena data yang dicari ada pada Sub Vektor1 dan sudah sama, maka prose pencarian selesai.

6.2.3. Pencarian Berurutan Berindex / Index Sequential Search

Metoda pencarian berurutan terindeks (indexed sequential search) adalah metoda lain dari beberapa metoda pencarian yang dapat menaikan efisiensi pencarian pada tabel yang sudah dalam keadaan urut. Dalam hal ini diperlukan tabel tambahan yang disebut dengan tabel index. Setiap elemen dalam tabel index berisi suatu kunci dan pointer yang menunjuk kerekaman tertentu dalam tabel yang sesuai dengan kunci tersebut. Elemen-elemen dalam tabel index juga harus diurutkan seperti halnya dalam tabel yang asli.

Salah satu cara untuk menyusun tabel index adalah sebagai berikut. Di bawah ini adalah suatu tabel yang menunjukan tabel daftar mahasiswa dari suatu universitas yang sudah diurutkan menurut nomor mahasiswanya.

No	No. Mhs	Nama Mahasiswa	Thn	Faklutas
1.	951001	Andi Usman Alim	95	Komputer Akuntansi
2.	951102	Bambang Aji	95	Teknik Informatika
3.	951110	Eko Pambudi	95	Teknik Informatika
4.	961201	Ahmad Tirto Sudiro	96	Manajemen Informatika
5.	961003	Nyai Anteh Bulan	96	Komputer Akuntansi
6.	961101	Dewi Indra Utama	96	Teknik Informatika
7.	961105	Metlati Putih	96	Teknik Informatika
8.	961207	Warda Palu	96	Manajemen Informatika
9.	971204	Kamino Bali	97	Manajemen Informatika
10.	971214	Watak Abdulrahman	97	Manajemen Informatika
11.	971204	Doel Muslimin	97	Manajemen Informatika
12.	981009	Christy Manusama	98	Komputer Akuntansi
13.	981113	Ujang Mursalim	98	Teknik Informatika
14.	981116	Dimas Adi Winata	98	Teknik Informatika
15.	981223	Rr. Adi Hendra Ningrum	98	Komputer Akuntansi

Dari gambar diatas, jika kita melihat pada nomor mahasiswa, kita bisa mendapatkan informasi yang tersembunyi. Dua digit paling depan menunjukan tahun angkatan seorang mahasiswa masuk ke universitas; dua digit ditengah menunjukkan kode fakultas, 10 adalah Komputer Akuntansi, 11 adalah Teknik Informatika dan 12 adalah Manajemen Informatika. Dua digit terakhir menunjukan nomor urutnya.

Dengan melihat gambar di atas, kita bisa menyimpan tabel di atas untuk diimplementasikan menggunakan suatu struktur data tertentu untuk mendukung pencarian berurutan terindex dengan beberapa cara. Cara pertama adalah dengan menggunakan tahun angkatan sebagai kunci. Cara kedua menggunakan kode fakultas untuk kuncinya. Gambar di bawah ini , menunjukan penyimpanan tabel di atas dengan tahun angkatan sebagai kunci.

Tabel Data

Tabel Index

95	**1**
96	**4**
97	**9**
98	**12**
	16

No. Mhs	Nama Mahasiswa	Faklutas
1001	Andi Usman Alim	Komputer Akuntansi
1102	Bambang Aji	Teknik Informatika
1110	Eko Pambudi	Teknik Informatika
1201	Ahmad Tirto Sudiro	Manajemen Informatika
1003	Nyai Anteh Bulan	Komputer Akuntansi
1101	Dewi Indra Utama	Teknik Informatika
1105	Metlati Putih	Teknik Informatika
1207	Warda Palu	Manajemen Informatika
1204	Kamino Bali	Manajemen Informatika
1214	Watak Abdulrahman	Manajemen Informatika
1204	Doel Muslimin	Manajemen Informatika
1009	Christy Manusama	Komputer Akuntansi
1113	Ujang Mursalim	Teknik Informatika
1116	Dimas Adi Winata	Teknik Informatika
1223	Rr. Adi Hendra Ningrum	Komputer Akuntansi

Setiap elemen dalam tabel index terdiri dari dua buah medan, medan pertama menunjukan tahun angkatan yang dimaksud, medan kedua berisi pointer yang menunjukan ke tabel data yang rekaman pertamanya berisi sambungan dari nomor mahasiswa yang tahunnya dinyatakan dalam medan pertama. Sebagai contoh, rekaman pertama dalam tabel index, pointernya bernilai 1; hal ini menunjukan bahwa 1 adalah nomor rekaman pertama dalam tabel data yang tahun angkatannya adalah 95. Dalam rekaman kedua dari tabel index, pointernya bernilai 4; hal ini menunjukan bahwa 4 adalah nomor rekaman pertama dalam tabel data yang tahun angkatannya adalah 96, dan seterusnya.

Dengan cara ini bisa dihitung mahasiswa yang masuk pada universitas tersebut setiap angkatannya. Sebagai contoh, banyaknya mahasiswa yang masuk pada tahun 95 adalah sebanyak pointer tahun 96 dikurangi dengan pointer tahun 95, yaitu sebanyak 3 mahasiswa (4-1). Untuk mengetahui banyaknya mahasiswa yang masuk pada tahun 97 diperlukan data sentinel, yang nilai pointernya apabila dikurangi dengan nilai pointer pada tahun 97 adalah tepat. Dalam contoh di atas, elemen terakhir dari tabel index berisi data sentinel yang dimaksud.

Program yang akan disajikan di bawah ini merupakan contoh pembentukan berkas berurutan berindex berdasarkan tahun angkatan. Dalam program tersebut, tipe data untuk tabel index dan berkas yang dimaksud dibuat berbeda (karena memang biasanya berbeda). Deklarasi tipe data untuk tabel index dan tabel data yang berisi data mahasiswa adalah sebagai berikut:

```
Type    { * Elemen tabel index * }
        Index = record
                    Kunci       : string [2];
                    Awal        : integer
                end;

        { * Elemen data * }
        Data = record
                    No_Mhs      : string [4];
             Nama, Fakultas     : string [20];
                    end;
           { * Tabel Index * }
        Li      = array [0..101] of Index;
        { * Tabel data * }
        Ld      = array [1..100] of Data;
```

Dalam deklarasi di atas, ukuran tabel index dibuat seperti pada contoh diatas, untuk menyediakan tempat bagi data sentinel yang diperlukan. Data yang akan dicoba, sebelumnya sudah disimpan dalam berkas teks yang diberi nama COBADATA.DAT.

Secara singkat program di bawah ini bisa dijelaskan sebagai berikut. Setelah data dibaca dari berkas COBADATA.DAT (yang sudah dalam keadaan urut), dibentuk tabel index

seperti dijelaskan diatas. Jika kita ingin mencari suatu nomor mahasiswa, terlebih dahulu nomor mahasiswa tersebut dipecah menjadi dua untai, yaitu untai nomor urut mahasiswa. Pencarian dimulai dengan mencari tahun angkatan pada tabel index. Jika ditemukan, selanjutnya diteruskan dengan mencari data yang dimaksud pada tabel data. Jika tahun angkatan yang dicari tidak ditemukan, pencarian langsung dihentikan dan tidak perlu mencari pada tabel data, karena dalam tabel data yang terbentuk sesungguhnya merupakan sambungan dari tabel index. Untuk pencarian pada tabel index dan tabel data dilakukan dengan cara pencarian berurutan.

Keuntungan yang segera bisa dilihat dari metode ini adalah bahwa pencarian dapat dipercepat karena jika index yang dicari tidak ditemukan, maka berarti data yang akan dicari memang tidak ada dalam tabel data. Tetapi jika indexnya ditemukan, maka hanya sebagian kecil dari tabel data yang perlu kita cari . Hal ini tentu akan mepercepat pencarian dibanding harus mencari pada keseluruhan tabel data.

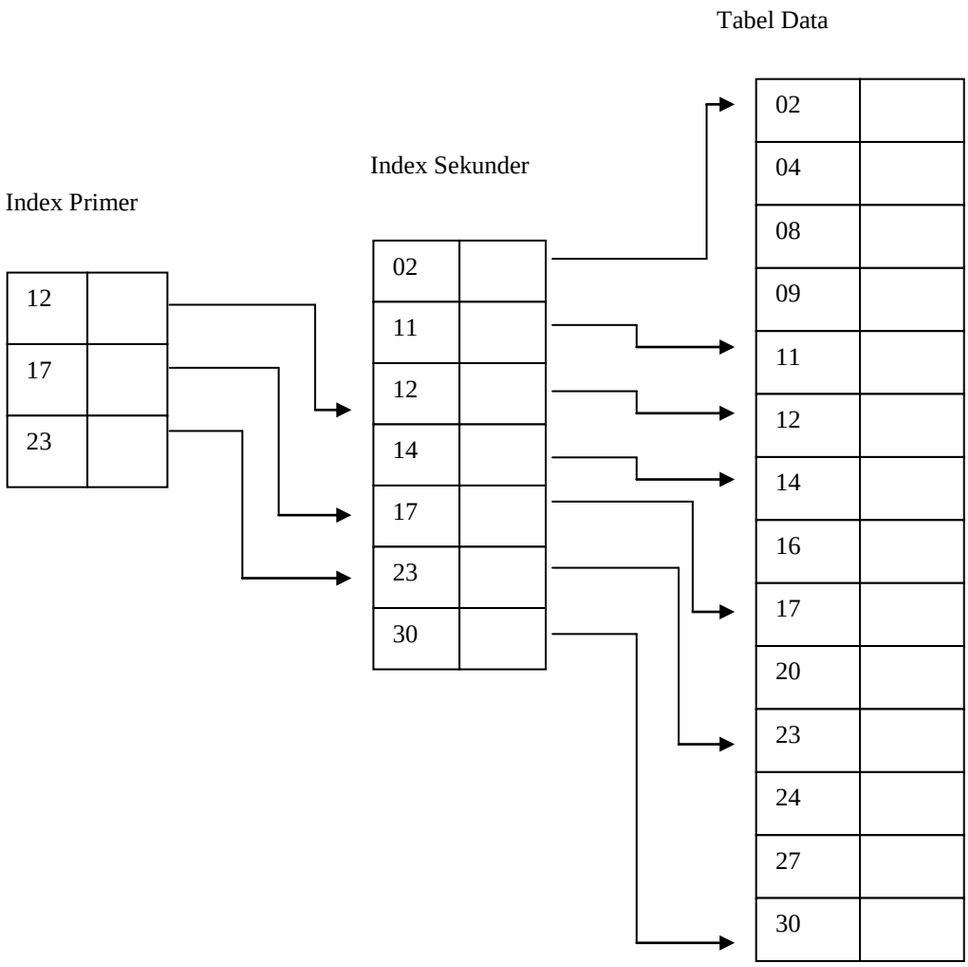
Pemakaian index juga bisa diimplementasikan menggunakan senarai/ link list seperti halnya dengan penggunanaan larik/array. Meskipun diperlukan ruang pengingat yang lebih banyak untuk pointer, tetapi yang melakukan penghapusan atau penyisipan akan lebih mudah dilaksanakan dibanding denga larik/array.

Jika tabel data yang digunakan cukup besar sehingga meskipun sudah digunakan tabel berurutan terindex efisiensi pencarian yang diinginkan tidak bisa dicapai (mungkin karena indexnya terlalu besar untuk mengurangi pencarian berurutan, atau karena indexnya tertentu kecil sehingga kunci-kunci yang berdekatan cukup jauh satu sama lain) seringkali digunakan index sekunder. Index sekunder ini juga berfungsi sama dengan index primer.

Penghapusan elemen dari tabel berurutan terindex akan lebih mudah apabila digunakan suatu tanda khusus (flag) yang menunjukan bahwa suatu elemen sudah dihapus. Dalam pencarian berurutan rekaman yang sudah diberi tanda akan diabaikan. Perhatian, bahwa jika ada suatu elemen dihapus, maka tabel index didiamkan saja, tetapi pada tabel data kita tambahkan tanda khusus pada elemen yang akan dihapus.

Penyisipan elemen ke dalam tabel berurutan terindex akan mengalami kesukaran jika tidak ada tempat yang tersedia baik pada tabel index ataupun pada tabel data. Kalaupun ada, juga harus melakukan penggeseran sejumlah elemen yang banyaknya tidak menentu. Tentu, jika ada elemen yang sudah ditandai untuk dihapus, mungkin hanya diperlukan sedikit penggeseran sebelum elemen yang sudah ditandai tersebut ditumpang tulisi oleh elemen yang baru.

Untuk implementasi menggunakan senarai berantai/ link list, penulis setahkan pada pembaca sekalian untuk mencobanya.


Tabel Data
Index Sekunder
Index Primer
12
17
23
02
11
12
14
17
23
30
02
04
08
09
11
12
14
16
17
20
23
24
27
30

BAB VII

FILE SEQUENTIAL (ARSIP BERUNTUN)

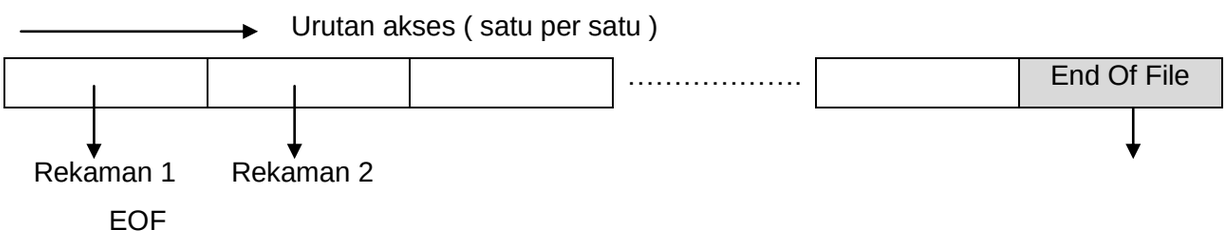
Memori utama dalam komputer tidak menyimpan data secara permanen. Apa yang tersimpan dalam memori akan hilang pada saat komputer dimatikan. Agar dat dapat di olah dikemudian hari, maka diperlukan memori yang dapat menyimpan data sehingga dapat diolah dikemudian hari. Memori ini dinamakan dengan meori sekunder (secondary storage). Media yang sudah dikenal untuk digunakan biasanya adalah Disk (disket, Harddisk, compact disk) dan tape.

Memori sekunder dapat menyimpan data dalam jumlah yang sangat besar. Selain itu, memori sekunder biasanya meyimpan data dalam bentuk arsip (**file**). Informasi yang disimpan dalam file dinamakan rekaman (**record**). Metode penyimpanan dan pengaksesan file bergantung pada metode pengorganisasian yang digunakannya. Beberapa metode organisasi file antara lain arsip beruntun (**sequential file**), arsip acak (**random file**), arsip berindeks (**indexed file**).

7.1. Definisi Arsip Beruntun

Arsip beruntun adalah sekumpulan rekaman yang disimpan di dalam media penyimpanan sekunder komputer yang dapat diakses secara berururtan mulai dari rekaman pertama sampai dengan rekaman yang terakhir secara berarah satu persatu.

Karena komputer tidak mengetahui akhir dari arsip, maka dibuatkan sebuah rekaman **fiktif** yang berfungsi sebagai tanda akhir dari sebuah arsip. Biasanya dinamakan sebagai **End Of File** (**EOF**).



7.2. Deklarasi Arsip Beruntun

Cara mendeklarsikan arsip dalam KAMUS adalah seperti di bawah ini :

KAMUS

Type Rekaman : record < deklarasi nama field dan tipenya >

Type Arsipberuntun : seqFile of Rekaman and EOF = <rekaman fiktif>

Contoh penggunaan arsip beruntun :

KAMUS

Type DataMhs : record < NIM : ineteger, Nama : string, IPK :real >

Type Arsipberuntun : seqFile of DataMhs and EOF = <9999,".",0.00>

MHS : ArsipMhs { Nama Arsip Mahasiswa }

KAMUS

Type BilBulat : integer

Type ArsipBil : seqFile of BilBulat and EOF = <9999>

BILANGAN : ArsipBil { Nama Arsip Bilangan Bulat}

Di bawah ini adalah ilustrasi dat dari deklarasi data di atas :

9940123	Citra Caraka	2,45
9940232	Ma'Ruf Manangsang	3,76
9940312	Rithamanubun	2,98
9940428	Karo Siahaya	3,43
9999999		0.00

a

355	34	866	4556	43	-45	756	3454	9999

b

7.3. Perintah Dasar untuk Arsip Beruntun

Setiap rekaman dapat diakses sesuai dengan urutannya dengan perintah pemanggilan akses yang ada. Perintah dasar untuk pengelolaan arsip beruntun adalah perintah primitif (berupa fungsi dan prosedur)yang telah dosediakan oleh **compiller**.

Beberapa perintah dasar yang digunakan untuk mengelola arsip beruntun adalah seperti di bawah ini :

1. Open

Fungsi : membuka arsip beruntun untuk siap dibaca. Pointer pembacaan menunjuk ke rekaman pertama.

Keterangan : perangkat keras yang berfungsi penting dalam operasi baca / tulis rekaman adalah **Head**. Posisi awal rekaman yang ditunjuk oleh **head** ditunjukan dengan **pointer**.

Contoh : OPEN(MHS,RekMhs) { MHS = nama arsip, RekMhs bertipe DataMhs }

OPEN(BILANGAN,I) { BILANGAN = nama arsip, I bertipe BilBulat }

Bila arsip yang dibuka berisi rekaman seperti pada ilustrasi data di atas, maka perintah open di atas akan menjadi :

RekMhs berisi <9940123,Citra Caraka,2,45>

I berisi 355

2. Read

Fungsi : membaca rekaman yang sekarang sedang ditunjuk oleh pointer pembacaan.

Contoh : READ(MHS,RekMhs)

READ (BILANGAN,I)

Bila pointer pembacaan menunjuk pada awal rekaman kedua dengan data seperti pada ilustrasi di atas, maka perintah read di atas akan menjadi :

RekMhs berisi <9940223,Ma'ruf Manangsang,3,76>

I berisi 34

3. Rewrite

Fungsi : menyiapkan arsip untuk perekaman

Contoh : REWRITE(MHS)

REWRITE (BILANGAN)

4. Write

Fungsi : Menulis rekaman ke dalam arsip

Contoh : WRITE(MHS, <9940123,Citra Caraka,2,45>)

WRITE (BILANGAN,<8000)

Catatan : Arsip yang dibuka untuk pembacaan (dengan perintah OPEN) tidak dapat digunakan untuk perekaman. Demikian juga sebaliknya, arsip yang dibuka untuk perekaman (dengan perintah REWRITE) tidak dapat dibaca. Operasi baca tulis tidak dapat dilakukan sekaligus pada arsip beruntun.

5. Close

Fungsi : menutup arsip yang telah digunakan untuk pembacaan ataupun perekaman.

Contoh : CLOSE(MHS)

CLOSE (BILANGAN)

7.4. Skema Pemroses Beruntun untuk Arsip Beruntun

Karena seluruh rekaman di dalam arsip tersusun secara beruntun, maka skema pemrosesan secara beruntun dapat diterapkan dalam pengelolaan arsip.

Contoh:

```
Procedure HIPK( output Natas2 : integer )
{ Menghitung jumlah mahasiswa yang IPK nya di atas 2, data dibaca dari arsip MHS }
Kamus Lokal
        RekMHS : DataMhs
```

```
ALGORITMA

OPEN(MHS, RekMHS) { RekMHS berisi rekaman pertama atau EOF }
IF RekMHS = OEF then { Arsip kosong }
        Natas2 ← 0
else
        Natas2 ← 0
        Repeat
                If RekMHS.IPK > 2.0 then
                                Natas2 ← Natas2  + 1
                Endif
                READ(MHS, RekMHS)  { baca rekaman selanjutnya}
        Until RekMHS = EOF
Endif
CLOSE(MHS)
```

7.4.1. Pemrosesan Teks

Teks adalah arsip yang terdiri atas deretan karakter. Untaian karakter tersebut dapat membentuk kata. Yang dimaksud dengan kata adalah kelompok karakter yang dipisahkan dengan kelompok lain oleh satu atau lebih spasi.

Pada implementasinya, ditambahkan *end of line* sebagai tanda dari akhir baris yang membedakan dengan baris lainnya.

Dalam meninjau model pengaksesan beruntun pada teks, pandanglah sebuah teks disusun oleh baris-baris teks. Andaiakan baris-baris teks tersebut dapat dipotong dan kemudian masing-masing ujung dari baris teks tersebut disambungkan, maka akhirnya didapatkan sebuah “PITA” yang berisi karakter-karakter.

Contoh

Saya adalah Mahasiswa STMIK Mardira Indonesia Jurusan Informatika yang sedang belajar Algoritma ……..

Jika suatu Teks didefinisikan dalam bagian DEKLARASI menjadi seperti :

```
DEKLARASI
        P : text ( P adalah variable Teks )
```

Pemrosesan teks yang utama adalah pembacaan. Pemrosesan teks adalah Pembacaan karakter secara beruntun dari awal hingga akhir. Akhir teks ditandai dengan karakter khusus (karakter titik (“.”)) yang dibedakan dengan karakter lainnya. Bila pembacaan bertemu dengan karakter khusus tersebut, maka pembacaan berhenti. Karakter khusus ini tidak dihitung sebagai unsur teks.

STMIK-Mardira Indonesia.

Teks yang terdiri dari 23 karakter. Akhir teks adalah ‘.’

.

Teks kosong

Didefinisikan RESET_TEKS adalah prosedur universal untuk teks. RESET_TEKS menyebabkan **pointer** akan menunjuk pada karakter pertama di teks. Jika teks kososng, **pointer** menunjuk titik. **Pointer** adalah identifikasi yang menunjuk pada karakter yang akan dibaca.

```
Procedure RESET_TEKS
{Menyiapkan Teks pada posisi awal }
{ K.Awal          : Sembarang }
{ K.Akhir         : pointer menunjuk pada karakter pertama di dalam teks
                    akibat pemanggilan prosedur ini, pointer menunjuk ke karakter pertama teks,
                    karakter yang ditunjuk mungkin '.'}
```

Notasi yang digunakan untuk membaca karakter dari teks P adalah :

```
DEKLARASI
C         : char ( karakter yang sedang di tunjuk oleh pointer baca }
P         : text ( P adalah variable Teks )

DESKRIPSI
Read (P,C)        { Membaca karakter yang ditunjuk oleh pointer baca dari teks P. Karakter yang dibaca
                    disimpan dalam peubah C. }
```

Contoh Kasus :

```
Procedure HITUNG_BANYAK_KARAKTER (Output : n : ineteger )
{Menghitung banyaknya karakter di dalam teks}
{ K.Awal          : Sembarang }
{ K.Akhir         : n berisi banyaknya karakter di dalam teks}

DEKLARASI
 ( Tidak ada )

DESKRIPSI
N ← 0
RESET_TEKS
Read (P,C)
While C ≠ '.' Do
        n ← n + 1
        Read (P,C)   { baca karakter berikutnya )
endwhile
```

```
Procedure ABAIKAN_SPASI
{Membaca deretan spasi dan mengabaikannya}
{ K.Awal          : Sembarang }
{ K.Akhir         : karakter bukan spasi ditemukan}

DEKLARASI
 ( Tidak ada )

DESKRIPSI
{Selama ada spasi dan belum titik, baca terus karakter di teks }
While (C ≠ '.') and (C ≠ ' ' ) Do
        Read (P,C)   { baca karakter berikutnya )
Endwhile
C ≠ '.' or C ≠ ' '
```

```
Procedure HITUNG_BANYAK_KATA (Output : nkata : ineteger )
{Menghitung banyaknya kata di dalam teks}
{ K.Awal          : nkata belum terdefinisi nilainya }
{ K.Akhir         : nkata berisi banyaknya kata di dalam teks}

DEKLARASI
 Procedure ABAIKAN_SPASI
{ Membaca deretan spasi dan mengabaikannya}

DESKRIPSI
RESET_TEKS
Read (P,C)
nkata ← 0
While C ≠ '.' Do
        ABAIKAN_SPASI

        { baca terus pita selama belum spasi atau belum titik }
        While ( C ≠ ' ' )  and ( C ≠ '.' ) Do
                Read (P,C)
        Endwhile
        ( C ≠ ' ' )  or ( C ≠ '.' ) Do
        nkata  ← nkata + 1
        ABAIKAN_SPASI
endwhile
```

7.5. Penggabungan Arsip

Pengganbunagn arsip (merging) dilakukan untuk menggabungkan rekaman yang disimpan di dalam dua buah arsip berbeda. Hasil penggabungan arsip disimpan dalam arsip baru.

7.5.1. Penggabungan Dua Buah Arsip dengan Penyambungan

Misalkan diberikan dua buah arsip bilangan bulat, arsip yang pertama diberi nama BILANGAN1 dan arsip yang kedua diberi nama BILANGAN2. Ilustrasinya seperti di bawah ini :

BILANGAN1

435	566	-456	856	8678	999

BILANGAN2

677	34	67	999

Apabila kedua arsip tersebut digabungkan dan hasilnya disimpan dalam arsip BILANGAN3, ilustrasi hasilnya adalah seperti di bawah ini :

BILANGAN3

435	566	-456	856	8678	677	34	67	999

Algoritmanya adalah seperti di bawah ini :

```
KAMUS
        Type BilBulat  : integer
        Type ArsipBil  : SeqFile of BilBulat and EOF = <999>

        BILANGAN1, BILANGAN2, BILANGAN3 : ArsipBil { nama arsip }

Procedure Penyambunganarsip1
{ Menggabungkan dua buah arsip dan hasilnya disimpan dalam arsip baru }

Kamus Lokal
        I : BilBulat

ALGORITMA
REWRITE(BILANGAN3)
OPEN(BILANGAN1, I )
While ( I <> 999 ) do
        WRITE(BILANGAN3, I )
        READ(BILANGAN1, I )
Endwhile
{ I = 999 }
{ Sambung BILANGAN3 dengan rekaman di BILANGAN2 }
While ( I <> 999 ) do
        WRITE(BILANGAN3, I )
        READ(BILANGAN2, I )
Endwhile
{ I = 999 }

WRITE(BILANGAN3, <999>)
CLOSE(BILANGAN1)
CLOSE(BILANGAN2)
CLOSE(BILANGAN3)
```

7.5.2. Penggabungan Dua Buah Arsip Terurut

Misalkan arsip Pertama dan arsip Kedua sudah terururt naik dan diinginkan arsip hasil penggabungan juga terurut naik.

Contoh :

BILANGAN1

43	56	59	85	98	999

BILANGAN2

16	34	67	999

Apabila kedua arsip tersebut digabungkan dan hasilnya disimpan dalam arsip BILANGAN3, ilustrasi hasilnya adalah seperti di bawah ini :

BILANGAN3

16	34	43	66	59	77	85	98	999

Algoritmanya adalah seperti di bawah ini :

```
KAMUS
        Type BilBulat  : integer
        Type ArsipBil  : SeqFile of BilBulat and EOF = <999>

        BILANGAN1, BILANGAN2, BILANGAN3 : ArsipBil { nama arsip }

Procedure Penyambunganarsip2
{ Menggabungkan dua buah arsip yang sudah terurt dan hasilnya disimpan dalam arsip baru
yang juga sudah terurut }

Kamus Lokal
        Angka1, Angka2 : BilBulat

ALGORITMA
OPEN(BILANGAN1, Angka1)
OPEN(BILANGAN2, Angka2)
REWRITE(BILANGAN3)

While ( Angka1<> 999 ) and ( Angka1<> 999 ) do
        If Angka1 ≤ Angka2 then
                WRITE(BILANGAN3, Angka1 )
                READ(BILANGAN1, Angka1 ) { Arsip BILANGAN1 maju satu rekaman }
        Else
                WRITE(BILANGAN3, Angka2 )
                READ(BILANGAN1, Angka2 ) { Arsip BILANGAN2 maju satu rekaman }
        Endif
Endwhile
{ Angka1  = 999  or Angka2  = 999}

{ Salin rekaman yang tersisa dari arsip BILANGAN1 dan BILANGAN2 }
While ( Angka1 <> 999 ) do
        WRITE(BILANGAN3, Angka1 )
        READ(BILANGAN1, Angka1 )
Endwhile
{ Angka1 = 999 }

While ( Angka2 <> 999 ) do
        WRITE(BILANGAN3, Angka2 )
        READ(BILANGAN2, Angka2 )
Endwhile
{ Angka1 = 999 }

WRITE(BILANGAN3, <999>)
CLOSE(BILANGAN1)
CLOSE(BILANGAN2)
CLOSE(BILANGAN3)
```

7.6. Pemutakhiran Arsip (Updating)

Pemutakhiran (Updating) adalah proses yang dilakukan untuk mengubah atau meremajakan rekaman arsip induk (master file). Peremajaan rekaman arsip dapat dilakukan dengan data rekaman yang baru di inputkan atau dibaca dari arsip transaksi.

BAB VIII

ALGORITMA DAN PEMROGRAMMAN TERSTRUKTUR

Konsep pemrogramman terstruktur memegang peran penting dalam merancang, menyusun, memelihara dan mengembangkan suatu program. Khususnya program aplikasi yang besar dan kompleks.

Konsep ini diungkapkan pertama kali oleh Prof. Edsger Djikstra dari Universitas Eindhoven pada tahun 1960-an. Prof. Edsger Djiktra mengungkapkan bahaya dari penggunaan instruksi loncatan pada program dalam segala bentuk pemrogramman. Oleh karena itu mulailah dikembangkan teknik pemrogramman terstruktur.

8.1. Istilah Dasar

Sebelum mempelajari pemrogramman terstruktur lebih lanjut ada beberapa istilah dasar yang perlu dipahami terlebih dahulu.

a. Program

Program adalah kata, ekspresi, pernyataan atau kombinasi yang disusun dan dirangkai menjadi satu kesatuan prosedur yang berupa urutan langkah untuk menyelesaikan masalah yang diimplementasikan dengan mengunakan bahasa pemrogramman tertentu sehingga dapat dieksekusi oleh komputer.

b. Bahasa Pemrogramman

Bahasa Pemrogramman adalah prosedur / tata cara penulisan program. Pada bahasa pemrogramman terdapat dua faktor penting, yaitu syntax dan semantik. Syntax adalah aturan-aturan gramatikal yang mengatur tata cara penulisan kata, ekspresi dan pernyataan, sedangkan semantik adalah aturan-aturan untuk menyatakan suatu arti.

c. Pemrogramman

Pemrogramman adalah proses mengimplementasikan urutan langkah untuk menyelesaikan suatu masalah dengan menggunakan suatu bahasa pemrogramman.

d. Pemrogramman Terstruktur

Pemrogramman Terstruktur merupakan proses mengimplementasikan urutan langkah untuk meyelesaikan masalah dalam bentuk program yang memiliki rancang bangun yang terstruktur dan tidak berbelit-belit sehingga mudah ditelusuri, dipahami dan dikembangkan oleh siapa saja.

8.2. Ciri Teknik Pemrogramman Terstruktur

Teknik pemrogramman terstruktur memiliki ciri-ciri atau karakteristik sbb :

a. mengandung teknik pemecahan masalah yang tepat dan benar
b. memiliki algoritma pemecahan masalah yang bersifat sederhana, standar dan efektif dalam memecahkan masalah
c. teknik penulisan program memiliki struktur logika yang benar dan mudah dipahami
d. program semata-mata terdiri dari tiga struktur dasar : sequence structure, selection structure dan Looping Structure.
e. menghindari penggunaan instruksi peralihan tanpa syarat tertentu yang menjadikan program tidak terstruktur dengan baik.
f. Membutuhkan biaya testing yang rendah
g. Memiliki dokumentasi yang baik
h. Membutuhkan biaya perawatan dan pengembangan yang rendah

8.3. Standar Program yang baik

Standar pemrogramman dibutuhkan untuk menciptakan suatu program yang baik dan memiliki portabilitas yang tinggi sehingga memudahkan dalam merancang dan merawat program serta meningkatkan efektifitas penggunaan peralatan komputer. Untuk menentukan standar program yang baik diperlukan beberapa standar sebagai dasar penilaian, seperti :

a) pemecahan masalah
b) penyusunan program
c) perawatan program
d) standar prosedur

Standar-standar tersebut sering dilihat oleh programmer sebagai batasan kreatiftas dan kemampuan untuk menuangkan berbagai ide ke dalam bentuk program. Namun dengan adanya standar, bagaimanapun juga akan membuat program menjadi konsisten dan mudah untuk dikembangkan.

8.3.1. Standar Teknik Pemecahan Masalah

Setelah masalahnya dipahami dengan baik, programmer tentu membutuhkan suatu teknik untuk memecahkan masalah tersebut, antara lain dikenal *Teknik Top Down* dan *Teknik Bottom Up*.

Teknik Top Down merupakan teknik pemecahan masalah yang paling umum digunakan. Pada teknik ini suatu masalah yang rumit dibagi-bagi ke dalam beberapa kelompok maslah yang lebih kecil. Dari kelompok masalah yang kecil tersebut dianalisis. Apabila memungkinkan, maka masalah tersebut dipilah lagi ke dalam bagian yang lebih kecil lagi

sampai tidak dapat dipilah lagi. Setelah pemilahan tersebut dilakukan sampai bagian yang terkecil, baru kemudian disusun langkah-langkah untuk menyelesaikannya secara detail.

Teknik Bottom Up merupakan pemecahan masalah yang mulai ditinggalkan dikarenakan susah untuk membuat standarisasi proses dari prosedur-prosedur yang sudah terbentuk yang akan digabungkan. Dalam teknik ini, jika ada permasalahan yang rumit, maka pemecahan masalahnya dilakukan dengan menggabungkan prosedur-prosedur yang ada menjadi satu kesatuan program guna menyelesaikan maslaah tersebut.

Proses dari masalah hingga terbentuk suatu algoritma disebut **tahap pemecahan masalah**, sedangkan tahap dari algoritma hingga terbentuk suatu solusi disebut dengan **tahap implementasi**.

8.3.2. Standar Penyusunan Program

Dalam menyusun program ada beberapa kriteria yang harus diperhatikan oleh programmer.

a. Kebenaran Logika dan Penulisan

Program yang ditulis harus mempunyai kebenaran logika pemecahan masakah maupun penulisan. Program harus mempunyai ketepatan, ketelitian dan kebenaran dalam perhitungan sehingga hasilnya dapat dipercaya.

b. Waktu Minimum Untuk Penulisan Program

Waktu minimum penulisan program adalah waktu yang harus tersedia secara wajar untuk menyusun program.

c. Kecepatan Maksimum Eksekusi Program

Beberapa faktor yang mempengaruhi kecepatan eksekusi program anatara lain bahasa yang digunakan, algoritma yang disusun, teknik pemrogramman yang diterapkan dan perangkat keras yang digunakan.

d. Ekspresi Penggunaan Memory

Seorang programmer perlu mempelajari teknik pembuatan program yang dapat meminimumkan penggunaan memori. Pemborosan penggunaan memori akan memperlambat kecepatan eksekusi program.

Untuk dapat meminimumkan penggunaan memori, maka perlu diperhatikan :

- Penggunaan tipe data yang cocok untuk kebutuhan pemrogramman.
- Menghindari looping untuk variable berindeks

e. Kemudahan Merawat dan mengembangkan Program

Program hendaknya memiliki struktur pemrogramman yang baik, struktur data yang jelas dan dilengkapi dengan dokumentasi sehingga mudah untuk dipahami dan dikembangkan.

f. User Friendly

Program yang disusun harus memiliki fasilitas-fasilitas yang memberikan kemudahan-kemudahan kepada user untuk mengoperasikannya.

g. Portability

Program yang disusun dapat dipakai dalam sistem operasi dan perangkat keras apa saja, sehingga fleksibel.

h. Pemrogramman Modular

Program terdiri dari modul-modul yang diusahakan tidak saling bergantungan satu sama lain. Namun demikian setelah dibangun dalam suatu sistem maka modul-modul tersebut menjadi satu kesatuan yang terintegrasi.

8.3.3. Standar Perawatan Program

Dalam menyusun suatu program, programmer harus mempertahankan faktor yang memudahkan dalam merawat dan mengembangkan program. Faktor tersebut antara lain :

a. Dokumentasi

Dokumentasi merupakan catatan dari setiap langkah pembuatan program dari awal sapai akhir. Dokumentasi berguna untuk menelusuri kesalahan jika ada ataupun untuk mengembangkannya.

b. Penulisan Program

Agar memudahkan proses perawatan program, maka sebaiknya penulisan program adalah sbb :

- Tulis satu instruksi dalam satu baris
- Pemisahan Modul Program dengan beberapa spasi
- Bedakan Penulisan Instruksi program, variable atau huruf kecil
- Bedakan tabulasi untuk bagian yang termasuk dalam bagian looping
- Hindari penggunaan kostanta yang mempunyai kemungkinan akan berubah
- Batasi jumlah baris dalam satu mosul

8.4. Menulis Program

Programmer adalah orang yang bekerja menyusun program. Untuk menghasilkan program yang baik diperlukan programmer yang baik dan berkualitas pula. Adapun kriteria programmer yang baik adalah :

a) mampu menyusun pemecahan masalah dengan baik
b) menguasai bahasa pemrogramman dengan baik
c) mampu menulis program dengan teknik pemrogramman yang baik
d) mampu menyusun program dengan baik
e) dapat bekerja sama dalam suatu tim
f) dapat bekerja secara efisien dan tepat waktu.

Pustaka

- Munir Rinaldi. Ir., Lidya Leoni. Ir., Algoritma dan Pemrogramman Buku 1, Informatika ITB, Bandung, 1997
- E. Horowitz and S. Sahni, Fundamentals of Computers Algorithms, Computer Science Press, 1978
- Wirth Niklaus, Algoritma + Struktur Data = Program, Andi, Yogyakarta, 1997
- Pranata Antony, Algoritma dan Pemrogramman, J&J Learning, Yogyakarta, 2002